Universidad Carlos III de Madrid

Escuela Politécnica Superior

Grado en Ingeniería Telemática

Análisis del control de topología en OF e implementación de algoritmo de enrutado

Trabajo Fin de Grado

Autor: Julián Merino Fresno
Tutor: Antonio de la Oliva

Septiembre de 2014

A mis padres.

Agradecimientos

A mis padres, hermana y abuelos, es a ellos a quien debo todo lo que soy.

A mi tíos: Willy, quien ha sido la fuente de motivación para elegir este camino, Javier, mi padrino, y con quien comparto grandes conversaciones y me hace sufrir encima de la bicicleta, y muy especialmente a mi tía Yolanda, hayá donde esté quiero agradecerle el interés que siempre tuvo por mi carrera universitaria y mi carrera profesional y todos los consejos que siempre me daba.

A Débora, siempre dispuesta a hacerme sonreír y llenarme de energía para poder afrontar cualquier situación.

A todas las personas con las que he compartido estos años de mi vida, y a aquellas que se han convertido en grandes amigos, Alberto, Lucia, Diego, Marga, Robi, Primo, Carlos, David, Juan, Martín y muchos otros que han estado ahí en los momentos en los que hacia falta un pequeño empujón para seguir hacia delante.

A mi tutor, Antonio por su supervisión y el tiempo dedicado a que este proyecto haya salido adelante.

Nothing great was ever achieved without enthusiasm.
Ralph Waldo Emerson(1803-1882)

Resumen

La evolución de las tecnologías de la información y la llegada de la era digital ha hecho que el tráfico en la red se haya disparado exponencialmente, haciendo que aparezcan limitaciones y problemas que las redes tradicionales no pueden abordar. Estas limitaciones y problemas hacen que la visión clásica del paradigma de la red cambie, dando más importancia al software y a la virtualización, y no tanto a grandes equipos hardware.

Una de las causas por las que se plantean cambios en las redes tradicionales es el aumento de densidad de usuarios y el tráfico. Este proyecto se centra concretamente en las redes inalámbricas. El proyecto europeo CROWD (Conectivity Management for eneRgy Optimised Wireless Dense networks) promueve un cambio en el paradigma de red centrando sus objetivos en la configuración dinámica de la red, un control energético eficiente y en mecanismos para la gestión de conexión.

CROWD se divide en cinco módulos: WP1, WP2, WP3, WP4 y WP5. El alcance de este Trabajo de Fin de Grado está dentro del módulo WP4, que es el módulo encargado de la gestión de la movilidad dentro de la red. Esta gestión se realiza mediante una arquitectura de una red basada en software, la cual cuenta con un plano de control distribuido encargado de realizar todas las tareas relativas a la gestión de la movilidad entre los nodos.

La separación del plano de control y el plano de datos que nos proponen las redes basadas en software hace que se planteen nuevos retos y funcionalidades. Aprovechando la visión general de la red que se tiene desde el plano de control en este tipo de red en este Trabajo de Fin de Grado se ha desarrollado un módulo en lenguaje Python el cual, ejecutado en el controlador, pueda obtener una representación lógica de la topología de la red a la que se encuentra conectado.

La otra parte de este proyecto se basa en probar una optimización de un algoritmo de enrutado. En una red Wireless basada en SDN que un usuario cambie de punto de acceso en la red puede hacer que el camino desde este punto de acceso hasta el gateway (u otro nodo) sea completamente diferente y obligue a realizar demasiados cambios en las tablas de flujo de los nodos. Esta optimización surge para intentar evitar, en la medida de lo posible, el mayor número de cambios en las tablas de flujos de los nodos de la red y se basa en obtener un camino desde un nodo a otro que más se parezca a un camino dado.

Este último módulo del proyecto se probará en diferentes SDN con diferentes tipos de topología y tamaños, y se comparará con el algoritmo de Dijkstra para intentar llegar a una conclusión en cuanto a la viabilidad de incluirlo como optimización en el módulo WP4 del proyecto CROWD.

Palabras clave:

CROWD, SDN, OpenFlow, Dijkstra, WP4.

Abstract

The evolution of information technologies and the advent of the digital age skyrocketed the network traffic, finding some limitations and problems that traditional networks can't address. These limitations and problems show the need of a change of the classic network paradgm, giving more importance to virtualization and software.

One of the reasons why this changes are arising in traditional network is the increase of user density and traffic. This project focuses specifically on wireless networks. The CROWD (Conectivity Management for eneRgy Optimised Wireless Dense networks) is a European project that promotes a paradigm change focusing the network changes in a dynamic configuration, an efficient energy control and management mechanisms for connection control.

CROWD is divided into five modules: WP1, WP2, WP3, WP4, WP5 and WP6. The scope of this project is within the WP4 module, which is responsible of the mobility management in the network. This is accomplished through a software defined network that has a distributed control plane responsible for conducting all task related to mobility management between nodes.

The separation of the control and data plane that software defined networks makes new challenges arise. Taking advantage of the view of the network that the control plane have I have developed a python module for one SDN controller which can obtain a logic representation of the network topology.

The other part of this project aims to develop and test a optimization of a routing algorithm. In a SDN based wireless network a hangover between two access points can make the path of that user to the network gateway very different of the old one, forcing too many changes in the flow tables of the network. This optimization arises to avoid, as much as possible, the greatest number of changes in the flow tables of the nodes in the network trying to obtain a path the similar as posible to the old one.

This last module of the project will be tested in different SDN networks with different sizes and topologies and it will be compared with Dijkstra algorithm trying to reach a conclusion of the viability of including it as a optimization for the WP4 module of the CROWD project.

Keywords:

CROWD, SDN, OpenFlow, Dijkstra, WP4.

Índice General

Lista de Figuras

Glosario

La lista de los acrónimos utilizados en este proyecto es la siguiente:

SDN Software Defined Networking

NFV Network Functions Virtualization

BFS Breadth-first search

OF OpenFlow

API Application Programming Interface

VxLAN Virtual Extensible Lan

ASIC Application Specific Integrated Circuit

CROWD Connectivity management for eneRgy Optimized Wireless dense networks

GW Gateway

CAPEX Capital Expenditure

OPEX Operating Expense

NAT Network Address Translation

VRRP Virtual Router Redundancy Protocol

REST API Representational State Transfer

WAN Wide Area Network

Parte I

Introducción

Capítulo 1

Introducción

1.1. Introducción

Las redes tradicionales están experimentando ciertas limitaciones y problemas debido al aumento exponencial del tráfico en ellas, por eso, están surgiendo nuevas estructuras y nuevos protocolos de red para intentar solucionarlos. El concepto de SDN [17] aparece como una nueva forma de construir y gestionar redes, desacoplando el plano de control y el plano de datos con el objetivo de optimizar cada uno de estos por separado.

OpenFlow [16] aparece de la mano del concepto de SDN y es considerado el primer estándar para *software defined networking*. Este estándar es el más conocido y utilizado para la comunicación entre el plano de control y el plano de datos de la arquitectura SDN y es el que usará nuestra plataforma de pruebas.

Este trabajo de fin de grado cuenta con dos objetivos principales. El primero, desarrollar un módulo para el controlador SDN Ryu que obtenga la topología física de la red y la represente de manera lógica para poder tener un control sobre ella. El segundo objetivo es desarrollar un algoritmo de enrutado de nivel 2 y estudiar sus prestaciones con respecto al algoritmo de Dijkstra.

1.2. Objetivos

- Control de la topología de una red SDN basada en OpenFlow mediante un módulo del controlador Ryu [12].
- Desarrollo y pruebas de algoritmo de enrutado de nivel 2.

1.3. Fases del desarrollo

El Trabajo Fin de Grado se dividió en las fases de desarrollo siguientes:

- **Documentación y análisis**: estudio de la documentación y análisis de la problemática

- **Desarrollo del control de topología**: desarrollo del código necesario para un módulo de control de la topología con el controlador Ryu
- **Desarrollo del algoritmo de enrutado**: desarrollo de un algoritmo de enrutado basado en BFS.
- **Toma de datos**: Toma de datos del rendimiento del algoritmo de enrutado.
- **Evaluación de los resultados**: con los resultados obtenidos, se analizaron y se compararon con las obtenidas en las diferentes pruebas para obtener una conclusión

1.4. Estructura de la memoria

La memoria se divide en varias partes, que a su vez se dividen en distintos capítulos. El contenido de cada parte y capítulo se resume a continuación:

1. **Primera parte: Introducción**. En esta parte se explican los objetivos, las fases del trabajo y la estructura de la memoria.
   - Capítulo 1. Introducción
2. **Segunda parte: Estado del arte**. Se explican los diferente conceptos y retos en los que se basa este trabajo.
   - Capítulo 2. Software Defined Networking y OpenFlow
3. **Tercera parte: Trabajo realizado**. En esta parte se explica el trabajo realizado para el desarrollo y pruebas.
   - Capítulo 3. Arquitectura del prototipo.
   - Capítulo 4. Despliegue del prototipo.
4. **Cuarta parte: Conclusiones y trabajos futuros**. En esta parte se explican las conclusiones obtenidas del trabajo y futuros proyectos para ampliar el actual. Sólo contiene un capítulo:
   - Capítulo 5. Conclusiones y trabajos futuros
5. **Quinta parte: Anexos**. En esta última parte se amplía la información de algunas partes del Trabajo Fin de Grado:
   - Apéndice A: Planificación de tareas y presupuesto. En este anexo se describen las tareas del proyecto y los costes de estas.
   - Apéndice B: Código completo del módulo de topología
   - Apéndice C: Código completo del algoritmo de enrutado
   - Apéndice D: Datos recogidos.

Parte II

Estado del Arte

Capítulo 2

Software Defined Networking y Openflow

2.1. SDN y OpenFlow

2.1.1. SDN

2.1.1.1. Introducción

SDN es un cambio del paradigma tradicional de la red. La principal característica es la separación del plano de datos del plano de control. Los equipos de red serán vistos como entidades del plano de datos gestionados desde un plano de control centralizado. El plano de control se localiza en el controlador SDN, donde se implementan todo tipo de reglas a ejecutar en el plano de datos.

2.1.1.2. Arquitectura de red tradicional

La arquitectura de red tradicional esta compuesta por elementos de conmutación que cuentan, cada uno de ellos, con un plano de control y un plano de datos. Esto hace que la gestión de la red se realice de manera distribuida, y dependa del firmware que cada fabricante instale en su conmutador. Las redes tradicionales cuentan con limitaciones y problemas conocidos, tales como:

- Dependencia de los grandes fabricantes de equipamiento.
- Difícil escalado. Debido a la dependencia anteriormente citada.
- Basadas en gran cantidad de reglas definidas.
- Grandes costes operacionales. Asignación de direcciones, configuración de listas de control de acceso, ect.
- Grandes costes de equipamiento. Equipos altamente configurables o de grandes prestaciones cuentan con un gran coste.

Estas limitaciones son las más generales que nos imponen las redes tradicionales y esto ha hecho que el paradigma de la red haya ido cambiando estos últimos años en la dirección de la virtualización y el software.

2.1.1.3. Arquitectura de SDN

Las redes definidas por software aparecen como una nueva forma de ver el diseño, construcción y creación de redes. La principal diferencia frente a las redes tradicionales es la separación del plano de control y el plano de datos, permitiendo optimizar cada uno de estos elementos por separado. Esta separación permite tener rápidamente una visión general de todo el entorno de red y realizar modificaciones y cambios sobre ella.

Como ya hemos comentado SDN cuenta con dos capas bien definidas: la capa de control y la capa de datos. La capa de datos se compone de switches, routers, hosts, medios de transmisión etc...La capa de control, sin embargo, se compone de un software que toma decisiones de todo tipo sobre la capa de datos a la que está conectada.

Estas dos capas se comunican entre sí con las llamadas *SouthBound APIs* que permite a la capa de control comunicarse con la capa de datos para realizar su gestión y configuración. Hay varias APIs que permiten la comunicación entre estas dos capas, siendo las más conocidas VxLAN y OpenFlow, hablaremos de esta última más adelante.

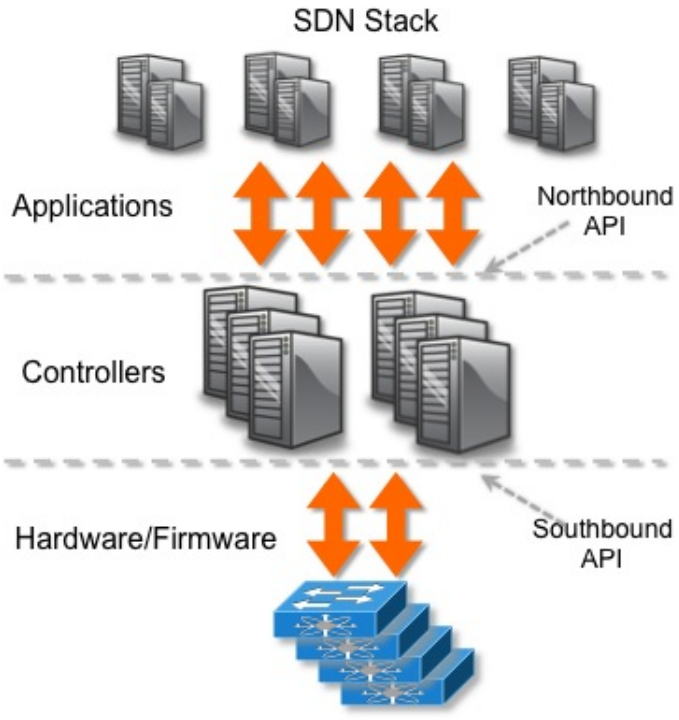

Figura 2.1: Esquema de arquitectura una red SDN [14]

Encima de la capa de control se ubica la capa de aplicación. Las *NorthBound APIs* resuelven el problema de la comunicación entre el plano de control y las aplicaciones, y es aquí donde aparecen verdaderos cambios con respecto las redes tradicionales. Las diferentes *NorthBound APIs* nos ofrecen gran flexibilidad para desarrollar y ejecutar aplicaciones en la red.

Esta arquitectura nos ofrece las siguientes ventajas frente a las redes tradicionales, estas son algunas de ellas:

- Reduce las inversiones en equipamiento. La flexibilidad que ofrece una red basada en software hace que se hagan innecesarias grandes inversiones en ASICs.
- Reduce los costes de operación. El control y gestión de la red es más fácil y está centralizado.
- Agilidad y flexibilidad. Se pueden desarrollar rápidamente aplicaciones para ir adaptándose a los cambios que sufren las redes.
- Aumenta el margen para innovación. Debido al gran API con el que cuenta.

2.1.2. OpenFlow

2.1.2.1. Introducción

Existe bastante confusión alrededor del termino OpenFlow, lo primero que se ha de tener claro es lo siguiente: SDN no es OpenFlow ni OpenFlow SDN. Openflow es uno de los distintos protocolos de comunicación entre el plano de control y el plano de forwarding, y está clasificado dentro de lo que se conoce como *Southbound API*.

El nacimiento de OpenFlow se remonta al año 2008 en la universidad de Standford, desde la cual se lanzó la versión 1.0 de este protocolo. Desde entonces se ha ido desarrollado el protocolo que actualmente se encuentra en su versión 1.3.4.

2.1.2.2. Switches OpenFlow

Los switches OpenFlow [20] son equipos que cuentan únicamente con un planos de datos y una conexión al controlador, del cual recibirán instrucciones para hacer modificaciones sobre sus tablas de flujos.

La estructura de un switch OpenFlow de manera gráfica sería la siguiente:

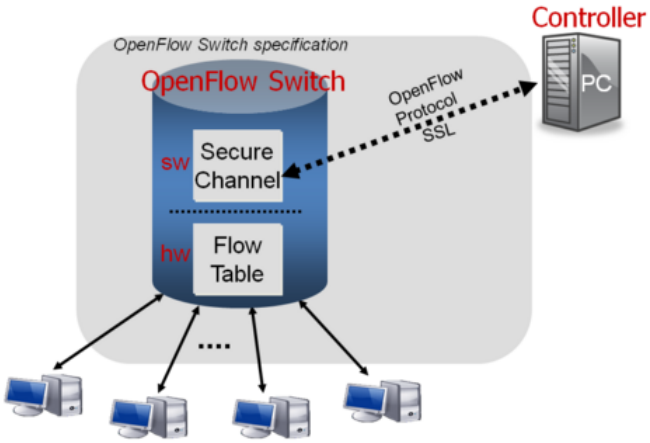

Figura 2.2: Arquitectura de un switch OpenFlow

Como hemos comentado los switches OpenFlow cuentan con varias tablas de flujos las cuales indican que tarea realizar con los paquetes que ingresan al switch. Estas tablas cuentan con los siguientes campos:

- Cabecera: Son los datos para hacer el *matching* con los paquetes a través del puerto por el que hayan ingresado y los datos que contengan en su cabecera.
- Prioridad: Da una prioridad a la regla para tomar decisiones en caso de que se puedan realizar varias acciones para un mismo paquete.
- Contador: Se actualiza cada vez que un paquete coincide con esa regla.
- Instrucciones: Acciones a realizar en esa regla.
- Timeout: Tiempo máximo que puede estar la entrada en la tabla sin recibir ninguna modificación. Una vez transcurrido este tiempo, la regla expirará.
- Cookie: No se usa para el filtrado de paquetes, pero se puede usar desde el controlador para llevar a cabo diferentes acciones, como la creación de estadísticas, modificar flujos ect...
- Flags: Son utilizados para modificar la forma de gestionar las entradas de la tabla.

Todos estos campos serán procesados por el switch en busca de coincidencias para realizar la acción correspondiente con los paquetes que le llegan. Cada vez que haya una coincidencia, apuntará las acción correspondiente en el *Action Set* y una vez se han evaluado todas las tablas, se procede a ejecutar, en orden, todas las acciones que se encuentren reflejadas en el *Action Set*.

Hay tres tipos de acciones básicas que todos los switches OpenFlow han de soportar:

- Reenviar el paquete: Una vez haya coincidencia se reenviará el paquete por el puerto indicado.
- Encapsular y reenviar al controlador: usualmente esto se realiza cuando se recibe un paquete de un nuevo flujo. El controlador decidirá la acción a realizar, que puede ser descartar el paquete, instalar un nuevo flujo en los switches o cualquier otro tipo de acción.
- Eliminar el paquete: Se elimina el paquete.

2.2. Movilidad en el proyecto CROWD

El proyecto europeo CROWD se centra en el estudio de la gestión de la movilidad para redes densas, para esto, el módulo WP4, el encargado de la gestión de la movilidad, basa su solución en el protocolo OpenFlow y en una red SDN. Esta solución nos ofrece un alto grado de escalabilidad y flexibilidad, lo que hace que sea posible desarrollar nuevos protocolos y optimizaciones sobre ella.

El plano de datos de esta red son puntos de acceso que se comportan como switches OpenFlow con una interfaz radio, por esto, cuentan con una tabla de flujos con la que

comparar los paquetes que reciben para realizar diferentes acciones y enviar ese paquete por un puerto en concreto o devolverlo al controlador. En plano de control gestiona todos los flujos instalados en los puntos de acceso de la red.

Cuando el controlador es consciente de que un evento de movilidad se está llevando a cabo tiene la capacidad de modificar las tablas de flujos de los switches correspondientes para enviar los paquetes a la nueva ubicación del terminal conectado a la red [19] . Es aquí donde surge la idea de minimizar esos cambios en las tablas de flujos de los switches de la red, modificando el algoritmo actual que decide en camino del GW hasta el punto de acceso, aprovechando la facilidad de implementar ese tipo de modificaciones en la arquitectura CROWD.

2.3. Entorno Socio-económico

Las redes basadas en software y la virtualización de funciones de red están cambiando el mercado tradicional de las empresas de tecnología y comunicaciones. Se puede observar una tendencia en la que el software cada vez tiene más importancia, y el desacoplo que hace SDN de el plano de control y el plano de datos hace que tenga un espacio muy claro en el mercado.

El que la industria se centre en el software abre el mercado a muchas empresas y marcas para desarrollar sus propias soluciones software, haciendo que aparezca una competencia cada vez mayor y no se dependa de un hardware extremadamente caro para poder prestar diferentes servicios.

Otro de los cambios que ofrecen las redes basadas en software en este entorno es el gran ahorro de recursos en cuanto a tareas de gestión y mantenimiento de red cotidianas, así como para nuevos despliegues. Esto hace que esos recursos se puedan emplear en otras acciones para aportar valor a la empresa o producto.

Se muestra a continuación una gráfica comparativa del CAPEX (*inversiones en bienes capitales*) y OPEX (*inversiones en operaciones*) en redes tradicionales y en redes basadas en software en las cuales se observa una reducción significativa de todos estos costes.

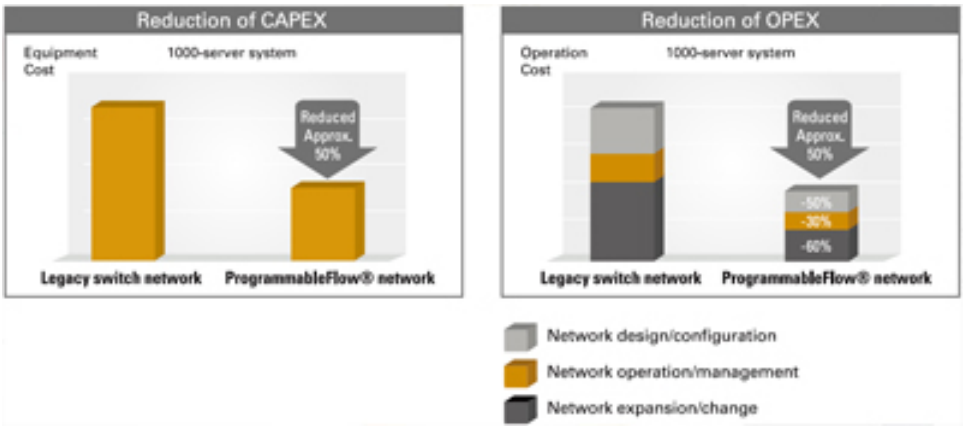

Figura 2.3: Reducción de costes con SDN [10]

Los datos mostrados en esta gráfica se han calculado para un datacenter que cuenta con 1000 servidores. Además de los costes reflejados se ha pasado de una necesidad de espacio

en rack de 32 U a 10 U y se ha reducido el consumo eléctrico de 14 Kw a 2.2 Kw.

Como conclusión, el entorno de las tecnologías de la información y comunicaciones está cambiando, tanto de puertas para dentro de la compañía, como de puertas hacia fuera. El desarrollo de nuevas funcionalidades y servicios se basará en la virtualización y en el desarrollo software.

2.4. Marco regulador

En este proyecto no aplica ningún marco regulador. El código del controlador SDN usado se distribuye libremente bajo una licencia Apache 2.0.

Parte III

Descripción del trabajo realizado

Capítulo 3

Arquitectura del prototipo

3.1. Introducción

En este capítulo se va a describir la arquitectura utilizada para el desarrollo y pruebas de los módulos desarrollados.

3.2. Objetivos

La arquitectura propuesta para la demostración tiene dos objetivos principales:

- Obtener la topología física de una red basada en OpenFlow utilizando el controlador Ryu.
- Desarrollar un algoritmo de enrutado de nivel 2 y probar su rendimiento.

3.3. Elementos del prototípo

El prototipo contará con los siguientes items.

- Controlador SDN: Será el nodo controlador de la red SDN. En él se ejecuta el código para obtener la topología y en él se puede ejecutar el algoritmo de enrutado desarrollado.
- Red SDN: Red de switches OpenFlow que será gestionada por el controlador. Sobre esta red se va a obtener la topología y sobre ella se va a poder ejecutar el algoritmo de enrutado.

El objetivo principal es obtener una plataforma robusta y de fácil gestión para la ejecución de todas las pruebas de rendimiento del algoritmo que se ha desarrollado. Todos los switches se conectarán al controlador y entre ellos y generarán eventos que nosotros utilizaremos para crear la topología.

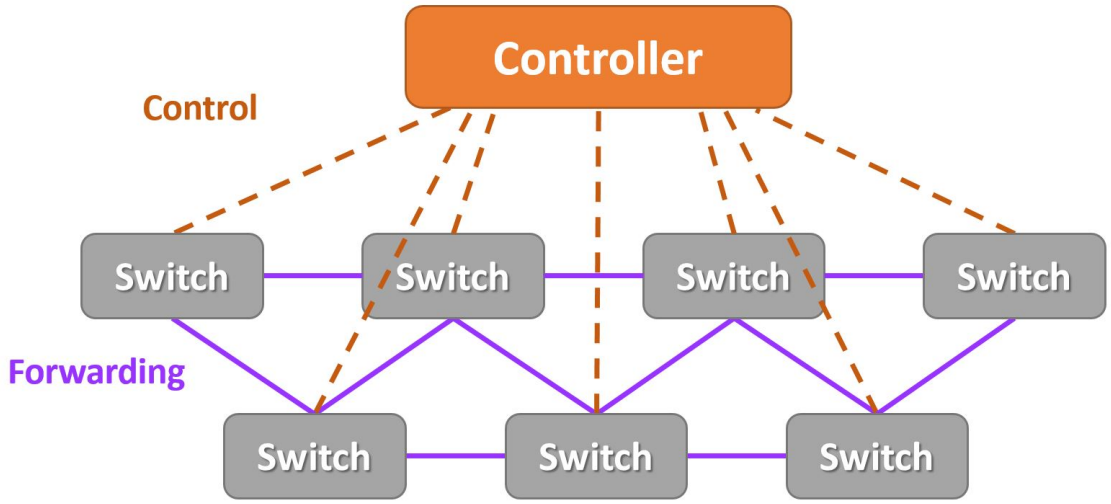

Figura 3.1: Esquema de conexión en una red SDN [17]

3.4. Plataforma de pruebas

La plataforma de pruebas se compone simplemente de un ordenador portátil. En él se ejecutará una máquina virtual Ubuntu con Mininet [3] y Ryu instalados donde generaremos las redes SDN basadas en OpenFlow y probaremos el código desarrollado.

Como elementos virtuales de nuestra topología de pruebas tendremos los siguientes elementos:

- Controlador: Es el *control plane* de nuestra red SDN. El controlador elegido será Ryu, en su version 3.4.
- Switches. Es el *data plane* de nuestra red SDN. Lo generaremos con mininet.

3.5. Alternativas de diseño

Para este Trabajo de Fin de Grado se ha optado por generar las redes con Mininet y usar Ryu de controlador SDN para esta.

Una de las posibles alternativas de diseño en cuanto a la definición del prototipo hubiese sido el elegir otro de los controladores SDN del mercado, como: POX, IRIS, MUL, NOX ect...

En cuanto a la red, se podría haber optado por un recurso hardware creando redes con equipos compatibles con OpenFlow, siendo esta una buena opción para la primera parte del proyecto y no tanto para la segunda, donde se necesitaban redes mucho más complejas para probar el algoritmo que se ha desarrollado.

Capítulo 4

Despligue del prototipo

4.1. Introducción

En esta sección se describirán los elementos empleados para esta demostración así como los elementos desarrollados específicamente.

4.2. Software empleado

4.2.1. Mininet

Mininet es un software que nos permite crear e interactuar con redes definidas por software virtuales en nuestro ordenador. Con esta aplicación hemos creado diferentes topologías con redes de switches Openflow para poder desarrollar y probar el proyecto. Esta herramienta se ha ejecutado sobre el software *VirtualBox* [15] en su versión 4.3.4.

Mininet se centra en diferentes atributos básicos:

- Es simple y rápida. Nos permite desplegar grandes redes en pocos segundos.
- Se pueden ejecutar programas reales. Cualquier programa que se pueda ejecutar en un sistema operativo Linux funciona con mininet.
- Facilidad de uso. Se pueden crear sencillos *scripts* en python para ejecutar mininet.
- Es open source. Se tiene acceso todo el código del programa.
- Está en continuo desarrollo.

A pesar de todo esto Mininet cuenta con algunas limitaciones a tener en cuenta, que no han sido impedimento para el desarrollo del este trabajo de fin de grado, pero que hay que mencionar, tales como:

- Si se precisan características adicionales hay que desarrollarlas para el controlador OpenFlow.
- No hace NAT de manera nativa, conlleva un cierto desarrollo el que lo haga.

- Mininet no tiene una buena noción del tiempo, esto hará que redes de gran velocidad (100 Gbps) no sean fáciles de emular.
- Se tiene que tener en cuenta que el sistema impondrá una capacidad máxima que tendrá que ser compartida entre todos los hosts.

Este recurso software se ha utilizado en base al tutorial que aparece en la página web de Openflow [5]. En este se explica de forma sencilla los pasos a seguir para instalar la maquina virtual y generar redes con mininet, así como utilizar un controlador externo para la red.

4.2.2. Ryu

Ryu es un componente de SDN completamente desarrollado en lenguaje python que soporta varios protocolos de comunicación, tales como: OpenFlow, Netconf, OF-Config etc... Se basa en dos pilares fundamentales: la agilidad y la flexibilidad.

Cuenta con muchos componentes diferenciados que podemos usar, juntar entre ellos y modificar para crear nuestra propia aplicación. Estos componentes ofrecen al usuario una interfaz de control, tienen estado y generan mensajes y eventos. Otro de los motivos que hacen a Ryu un framework muy ágil y flexible es la capacidad de poder crear componentes en otros lenguajes y juntarlos con componentes propios de Ryu escritos en python realizando pequeñas modificaciones, ya que basa el transito de información de los componentes en mensajes y no en referencias. Algunos de los componentes que incluye Ryu por defecto son:

- Compatible con OpenFlow 1.0 [6], 1.1 [7], 1.2 [8] y 1.3 [9].
- VRRP. Proporciona el protocolo VRRP compatible con OF.
- OF REST. Se pueden configurar switches a través del REST API

4.2.3. Networkx

Networkx [4] es una librería de python diseñada para trabajar con grafos y redes. Esta librería en particular me ha sido extremadamente útil a la hora de desarrollar todo el proyecto, ya que he utilizado numerosos recursos de ella. En primer lugar, gracias a esta librería, se hace muy sencillo el representar topologías de red ya que sus dos principales objetos son, los nodos y los vertices (nodes y edges, respectivamente). También existen dentro de esta librería generadores de topologías aleatorias, basándose en diferentes funciones de probabilidad, que nos han sido muy útiles a la hora de probar el software que hemos desarrollado en diferentes casuísticas para así comprobar su correcto funcionamiento.

4.3. Software desarrollado

En esta sección vamos a explicar de forma detallada cómo se han desarrollado los dos módulos principales que componen este proyecto, los problemas encontrados y las decisiones de diseño que se han tomado. No debemos olvidar que el objetivo principal de este proyecto

es probar el funcionamiento de un algoritmo de enrutado a nivel 2, y el obtener la topología de una red, obtener una estructura de datos lógica que la represente y probar el algoritmo sobre ella han sido los pasos a seguir.

4.3.1. Módulo topology

El objetivo del desarrollo de este módulo ha sido el desarrollo de un código que, mediante las herramientas previamente descritas, obtenga una representación lógica de una red. Esta ha sido la parte más complicada de todo el trabajo, no por la dificultad ha tenido, sino por la falta de documentación e información de Ryu. Lo que se necesita para construir una topología son: nodos y enlaces. Teniendo claro esto, el primer reto de este proyecto fue tener claro cómo identifica Ryu estos elementos y cómo representarlos en una estructura lógica coherente.

El primer punto lo resolví realizando los siguientes pasos:

- Leyendo la documentación de Ryu.
- Suscribiéndome al *mailing-list* [13]
- Haciendo debug del código de ejemplo de Ryu.

El segundo punto se resolvió realizando una pequeña búsqueda en la red y eligiendo la libería Networkx debido a la facilidad y recursos que me ofrecía.

Después de tener claro todo lo anterior me encontraba en un punto donde tenía claro como iba a representar mi topología y tenia en mente dos eventos clave que generaba Ryu cada vez que un nodo o un enlace se agregaban a la red. Estos dos eventos son *EventDP* y *EventLinkAdd*.

Atendiendo a la definición que nos ofrece la documentación de Ryu sobre estos eventos:

- *EventDP* evento para notificar la conexión/desconexión de un *switch*.
- *EventLinkAdd* evento para notificar la creación de un enlace entre dos *switches*.

La estructura del evento *EventDP* cuenta con dos elementos, *ev.dp.id* y *ev.enter*. El primero nos indica un identificador único para el switch que se está añadiendo a la topología y es el identificador que usaremos para crear nuestro nodo en la estructura de *Networkx*. El segundo elemento es de tipo de dato booleano y nos servirá para saber cuando este switch se añade o se elimina de la topología.

El segundo evento, *EventLinkAdd* cuenta con dos elementos, *ev.link.src* y *ev.link.dst*, que son dos tuplas que cuentan cada una con la siguiente estructura:

- *ev.link.src* cuenta con los siguientes datos (*srcdpid, srcport*).
- *ev.link.dst* cuenta con los siguientes datos (*dstdpid, dstport*).

Como se puede observar son cada elemento cuenta con el *datapath* destino u origen del link, y el puerto destino u origen del link.

Una vez recopilada toda esta información pude centrar los esfuerzos en puntos más concretos, centrando la atención en lograr capturar los eventos que hemos comentado con anterioridad para poder construir nuestra topología. Como código base utilizaremos un módulo que nos proporciona Ryu llamado *simple switch*, aprovechando la gestión de flujos de paquetes de datos que tiene por defecto este módulo para desarrollar solo el código es necesario.

Para añadir nodos y enlaces a un grado de *Networkx* necesitamos crearlo vacío, importaremos las librerías necesarias, y crearemos el gráfo de la siguiente manera:

```
import networkx as nx
(...)
self.topo = nx.Graph()
```

Para generar los nodos de la topología, como he comentado con anterioridad, capturé los eventos *EventDP*, y simplemente comprobando el valor del campo *ev.enter* añadiremos o borraremos el nodo de la topología. Esto lo haremos de la siguiente manera:

```
if ev.dp.id is None:
return
if ev.enter:
self.topo.add_node(ev.dp.id)
else:
self.topo.remove_node(ev.dp.id)
```

Por último para generar los enlaces de la topología, como he comentado con anterioridad, capture los eventos *EventLinkAdd* y creé el enlace en la topología desde la fuente al destino que apareciesen en ese evento. Esto se hizo de la siguiente manera:

```
src = ev.link.src
dst = ev.link.dst
if src.dpid is None or dst.dpid is None:
return
self.topo.add_edge(src.dpid, dst.dpid)
```

A continuación se muestra un ejemplo práctico del funcionamiento de este módulo. En esta primera figura se muestran los comandos necesarios para generar la red en Mininet. Para este ejemplo se decidió aprovechar un generador de redes con forma de árbol que tiene Mininet por defecto, al cual simplemente hay que indicarle la profundidad, y que el controlador es externo, ya que será este último el que ejecute el código desarrollado.

```
mininet@mininet-vm:~$ sudo mn --topo tree,3  --controller=remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(h1, s3) (h2, s3) (h3, s4) (h4, s4) (h5, s6) (h6, s6) (h7, s7) (h8, s7) (s1, s2) (s1, s5) (s2, s3) (s2, s4) (s5, s6) (s5, s7)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7
*** Starting CLI:
mininet>
```

Figura 4.1: Comando en Mininet para generar una red árbol de profundidad 3

Una vez ejecutado este comando, nuestro esquema de red sería el siguiente:

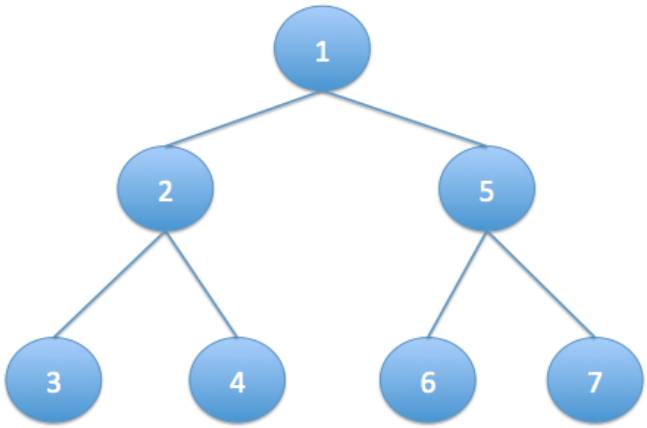

Figura 4.2: Topología de red generada

Sobre esta red ejecutaremos nuestro módulo del controlador Ryu, obteniendo la siguiente salida:

```
mininet@mininet-vm:~/ryu$ sudo PYTHONPATH=. ./bin/ryu-manager --observe-links ryu/app/myapp_BFS_v3.py
loading app ryu/app/myapp_BFS_v3.py
loading app ryu.controller.ofp_handler
loading app ryu.controller.ofp_handler
instantiating app None of Switches
creating context switches
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu/app/myapp_BFS_v3.py of GraphExplorer
(2293) wsgi starting up on http://0.0.0.0:8080/
Switch 1 anadido a la topologia
Switch 4 anadido a la topologia
Switch 2 anadido a la topologia
Switch 5 anadido a la topologia
Switch 7 anadido a la topologia
Switch 3 anadido a la topologia
Switch 6 anadido a la topologia
Edge, la topologia hasta el momento es: [(2, 4)]
Edge, la topologia hasta el momento es: [(1, 2), (2, 4)]
Edge, la topologia hasta el momento es: [(1, 2), (2, 4)]
Edge, la topologia hasta el momento es: [(1, 2), (2, 4)]
Edge, la topologia hasta el momento es: [(1, 2), (2, 4), (5, 7)]
Edge, la topologia hasta el momento es: [(1, 2), (2, 3), (2, 4), (5, 7)]
Edge, la topologia hasta el momento es: [(1, 2), (2, 3), (2, 4), (5, 6), (5, 7)]
Edge, la topologia hasta el momento es: [(1, 2), (2, 3), (2, 4), (5, 6), (5, 7)]
Edge, la topologia hasta el momento es: [(1, 2), (1, 5), (2, 3), (2, 4), (5, 6), (5, 7)]
Edge, la topologia hasta el momento es: [(1, 2), (1, 5), (2, 3), (2, 4), (5, 6), (5, 7)]
Edge, la topologia hasta el momento es: [(1, 2), (1, 5), (2, 3), (2, 4), (5, 6), (5, 7)]
```

Figura 4.3: Ejecución del módulo topology

4.3.2. Algoritmo de enrutado de nivel 2

Este algoritmo se basa en obtener un camino más parecido a uno dado, en este punto hablaremos de su desarrollo. Mi trabajo en este caso ha sido la traducción a lenguaje *python* de este algoritmo que estaba escrito en lenguaje matemático por Shmuel Zacks [**?**, 4] A continuación se muestra el código a traducir:

$d(s) = 0, \;\; d(v) = -1 \;\; \forall v \neq s$
$f_1(s) = 1, \;\; f_1(v) = 0 \;\; \forall v \neq s$
$i = 0$
while $d(b) = 0$
begin
$\forall x \in V$, **for which** $d(x) = -1$ **and** $\exists (v, x) \in E$ **with** $d(v) = i$ **do**
begin
$d(x) = i + 1$
let $V_x = \{v | (v, x) \in E, d(v) = i \}$
$M = max\{f_1(v) | v \in V_x\}$
if $x \in P$ **then** $f_1(x) = max(f_1(x), M + 1)$
else $f_1(x) = max(f_1(x), M)$
endif
enddo
$i \leftarrow i + 1$
endwhile

Figura 4.4: Algoritmo original en lenguaje matemático

Este algoritmo basado en BFS tiene una condición, siendo s el nodo origen, y v y b dos nodos, se tiene que cumplir que *dist(s, v)* $\leq$ *dist(s, b)*.

Una vez entendido el algoritmo, se dividió el desarrollo en dos bloques:

- *Actualización de pesos*: En este bloque se recorría la red asignado un peso a cada nodo. Este peso indicaba en cuantos nodos era igual el camino más parecido al original para llegar hasta él.
- *Obtención del nuevo Path*: Una vez actualizado el grafo de la red con todos los pesos, se recorre la red desde el último nodo hasta el nodo origen decidiendo en cada salto según el peso que tuviese el nodo siguiente, eligiendo el que tuviese mayor valor, es decir, el que fuese más parecido al path original.

El primer bloque se resolvió creando dos diccionarios en Python, una para almacenar

los pesos y otra para almacenar las distancias. Se utilizó un recurso de *Networkx* que nos permite mediante una sola línea de código obtener una lista con todos los nodos correspondientes al grafo.

```
distancia = dict(G.nodes(data=True));
pesos = dict(G.nodes(data=True));
```

Una vez se tuvieron las dos estructuras de datos se procedió a iterar sobre ellas para actualizar sus valores. Primero se obtiene un camino según el algoritmo de Dijkstra y en base a él se actualizan los valores de los pesos y las distancias. Una vez se obtiene este camino, se procede iterativamente comparando con los nodos que están a distancias n y $n+1$ y actualizando el valor al mayor posible. En todos estos pasos se comprueba la conexión entre los nodos a distancia n y $n+1$ y solo se aplica el algoritmo en los que aplique.

Habiendo actualizado con éxito todo el diccionario de pesos, el último paso para obtener un camino fue proceder iterativamente desde el nodo destino hasta el nodo origen eligiendo en cada iteración, de todos los nodos conectados, el nodo con mayor peso.

En la siguiente figura se puede observar esquemáticamente la ejecución del algoritmo. Se ha separado en tres pasos importantes que se describen a continuación.

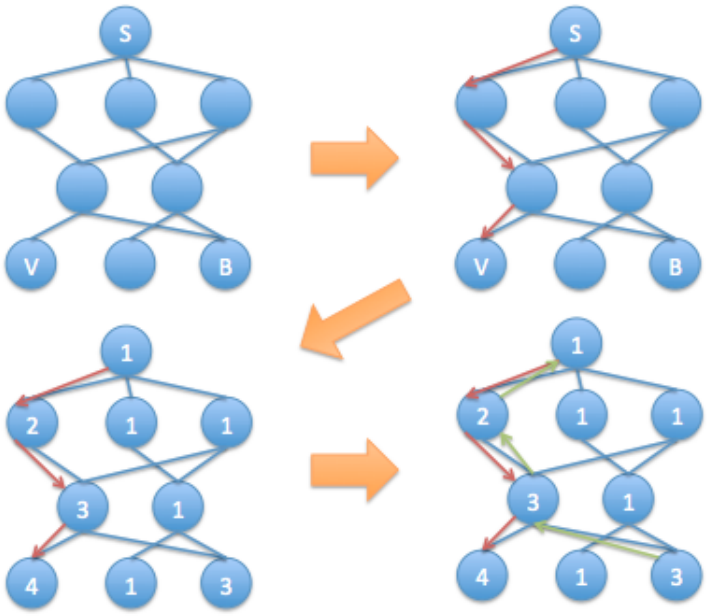

Figura 4.5: Esquema de funcionamiento del algoritmo

- Calculo del path con el algoritmo de Dijkstra con el que comparar desde el nodo S al nodo V
- Se dará un peso a cada nodo. Este peso se calculará según el máximo número de nodos del path original que se atraviesan para llegar hasta él.
- Cálculo del path mediante el nuevo algoritmo. Desde el nodo destino B se recorrerá el árbol hasta el origen S, eligiendo en cada iteración el nodo con mayor peso. Es decir, el nodo al que se puede llegar por un camino más parecido al original.

4.3.3. Alternativas de diseño

Debido a la facilidad de implementación se ha elegido Networkx para trabajar con los grados de red, pero una alternativa de diseño valida para la realización de este módulo, como del anterior, habría pasado por crear estructuras de datos manualmente sin utilizar las que Networkx nos ofrece.

4.4. Toma de datos y resultados

En esta sección se va a describir los procedimientos para la toma de datos y se mostrarán los datos obtenidos. En la toma de datos fijamos la atención en dos parámetros:

- Índice de mallado: Índice calculado dividiendo el número de enlaces de la red entre el número de nodos.
- Porcentaje de nodos que cambian con respecto al algoritmo de Dijkstra.

Teniendo en cuenta estos dos datos, se tomaron treinta muestras para redes de 150, 200, 250 y 300 nodos. Estas redes han sido generadas aleatoriamente gracias a un recurso de *Networkx* llamando *gnp_ random_ graph(n,p)* [2] donde n y p son:

- n: Número de nodos del gráfo.
- p: Probabilidad de crear un enlace entre dos nodos cualesquiera.

Para cada red se han tomado las 30 muestras ya comentadas y se ha calculado el valor medio. La metodología a seguir para el cálculo de estas ha sido la siguiente:

- 1 - Cálculo del camino entre dos nodos aleatorios a través del algoritmo de Dijkstra.
- 2 - Cálculo del camino entre el mismo nodo origen que en el camino anterior y otro nodo aleatorio a través del algoritmo de Dijkstra.
- 3 - Cálculo del camino entre el mismo nodo origen y el mismo nodo destino del paso 2 utilizando el algoritmo desarrollado.

Esta toma de datos se ha realizado teniendo en cuenta la limitación impuesta por el algoritmo desarrollado, que es: textitdist(s, v) $\leq$ dist(s, b), siendo s el nodo origen, v el primer nodo destino (punto 1.) y b el nodo destino de los puntos 2 y 3.

A continuación se mostrarán las gráficas en las que se representan todos los datos obtenidos. En todas ellas el eje de abscisas representa el indice de mallado para cada caso y el eje de ordenadas el porcentaje de nodos que cambian con respecto al algoritmo de Dijkstra.

Los diferentes indices dependen la probabilidad de crear dos enlaces cualesquiera, para cada red se han utilizado cinco probabilidades diferentes: 0.025, 0.0375, 0.05, 0.0725 y 0.1.

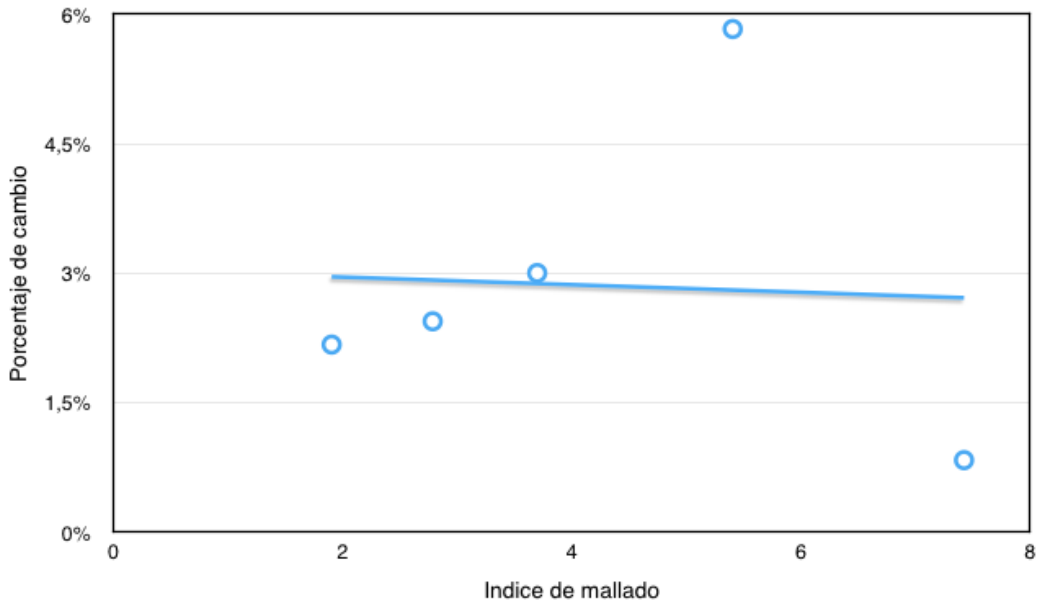

Figura 4.6: Gráfica para red de 150 nodos

Como se puede observar hay una tendencia a que nuestro algoritmo y el algoritmo de Dijkstra sean más parecidos cuanto más mallada sea la red (lo que hará que el camino entre dos nodos sea más corto para un mismo número de nodos).

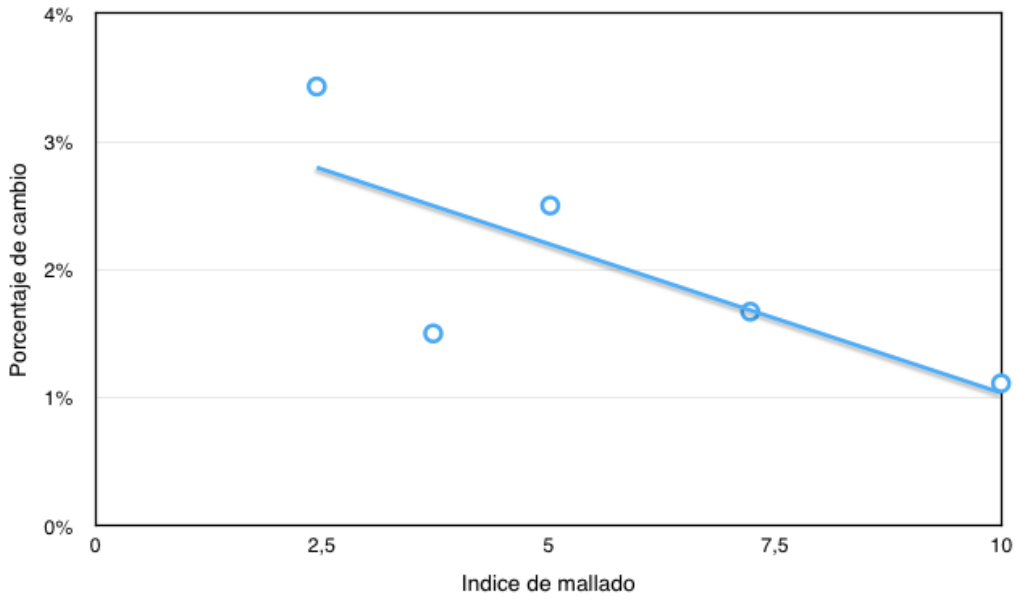

Figura 4.7: Gráfica para red de 200 nodos

Al igual que para una red de 150 nodos, se observa una tendencia a que nuestro y algoritmo y Dijkstra se parezcan más cuanto mayor sea el mallado de la red.

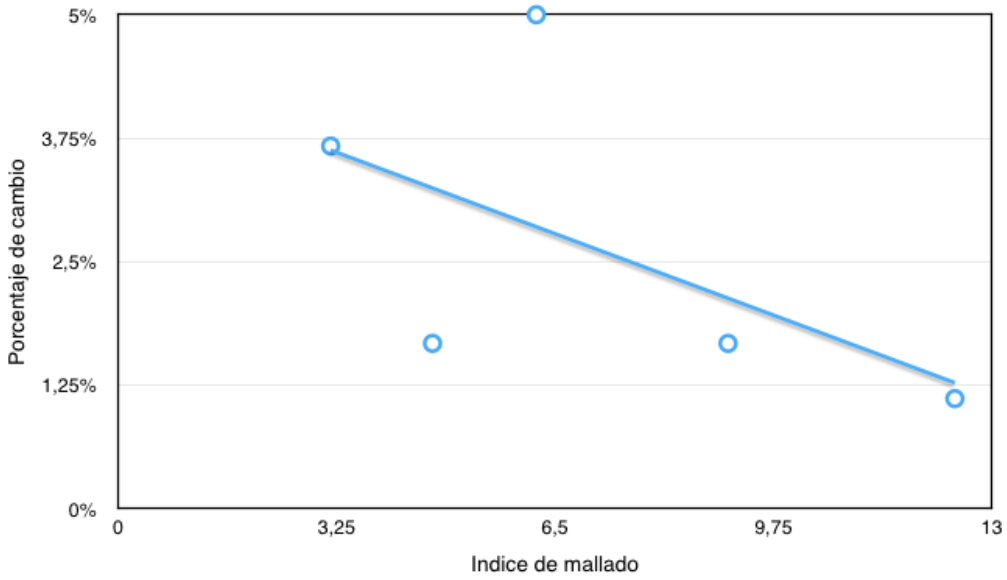

Figura 4.8: Gráfica para red de 250 nodos

Observamos el mismo comportamiento que para las redes de 150 y 200 nodos.

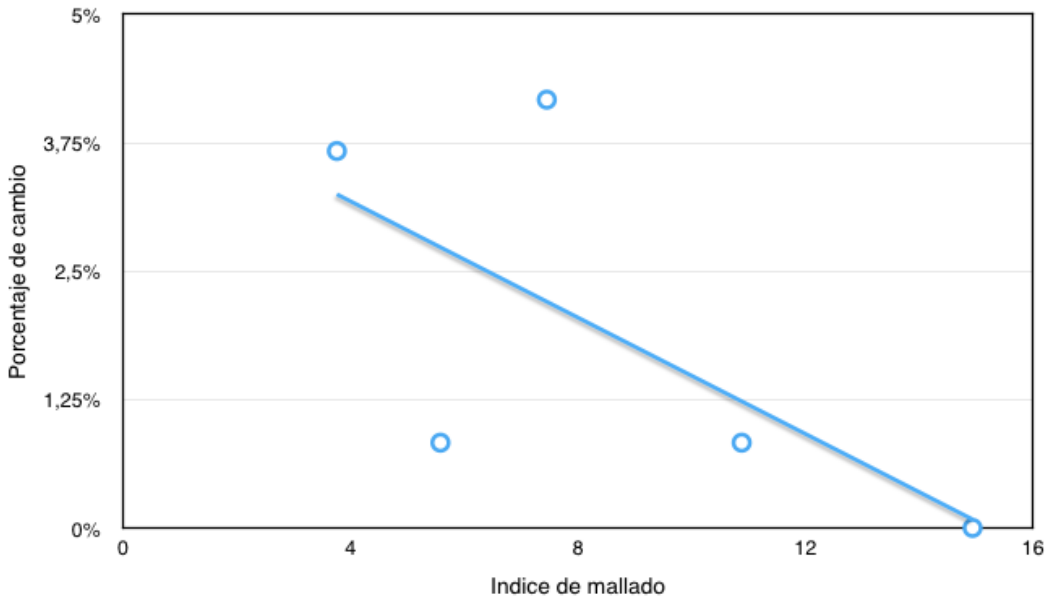

Figura 4.9: Gráfica para red de 300 nodos

Como en las tres gráficas anteriores la tendencia es la misma.

Parte IV

Conclusiones y trabajos futuros

Capítulo 5

Conclusiones y trabajos futuros

5.1. Conclusiones

Los resultados obtenidos de los dos módulos de este proyecto nos han permitido llegar a unas conclusiones claras con respecto a los dos. En primer lugar, en cuanto al módulo de control de la topología, esta visión general de la red que se tiene desde el controlador de una red SDN nos ha permitido obtener la topología de red con cierta facilidad y sobre ella ha sido realizada la toma de datos de rendimiento del algoritmo desarrollado.

En segundo lugar, con respecto al módulo en el cual desarrollamos un algoritmo de enrutado, viendo los datos, llegamos a la conclusión de que no merece la pena el gasto de recursos que supondría instalarlo en la red. Observando las gráficas y datos obtenidos sobre el rendimiento de este algoritmo podemos justificar esta conclusión de dos maneras, en primer lugar, la diferencia entre un algoritmo tradicional y el desarrollado no es significativa. En segundo lugar, se puede observar una tendencia a una mayor eficiencia del algoritmo desarrollado cuando el número de nodos aumenta, pero, las grandes redes SDN actuales (la red WAN de Google, por ejemplo) no cuentan con gran cantidad de nodos, lo que hace que este dato sea irrelevante, de momento.

Por lo tanto, la implementación de este algoritmo de enrutado en la red del proyecto CROWD no es viable, al menos de momento, mientras que las redes SDN no cuenten con mayor numero de nodos.

5.2. Trabajos futuros

Si las redes basadas en software comienzan a ser utilizadas en redes con mayor número de nodos y con un índice de mallado relativamente bajo sería interesante repetir este mismo experimento para ver que resultados nos ofrece, y sería interesante plantear una mejora del algoritmo para poder calcular caminos más largos que con el camino que se compara.

Como mejoras más generales sería interesante desarrollar una aplicación para automatizar las tareas de mantenimiento y gestión de una red pequeña tomando como base el controlador Ryu, ya que durante la evolución del proyecto ha ido actualizándose y ahora cuenta con más y mejores recursos. También podría utilizarse la visión general de la topología para crear una plataforma de gestión y control de la red, semejante a

las que ofrecen marcas como Cisco o Solarwins, de manera gratuita para pequeñas redes empresariales.

Bibliografía

[1] CROWD. http://www.networks.imdea.org/es/investigacion/proyectos/crowd, última vez visitada en Septiembre '14.

[2] gnp random graph(n,p). http://networkx.github.io/documentation/networkx-1.7/reference/generated/networkx.generators.random_graphs.gnp_random_graph.html#networkx.generators.random_graphs.gnp_random_graph, última vez visitada en Agosto '14.

[3] Mininet. http://mininet.org, última vez visitada en Agosto '14.

[4] Networkx. http://networkx.github.io, última vez visitada en Agosto '14.

[5] OpenFlow Tutorial. http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial, última vez visitada en Febrero '14.

[6] Openflow v1.0. http://archive.openflow.org/wk/index.php/OpenFlow_v1.0, última vez visitada en Agosto '14.

[7] Openflow v1.1. http://archive.openflow.org/wk/index.php/OpenFlow_v1.1, última vez visitada en Agosto '14.

[8] Openflow v1.2. http://archive.openflow.org/wk/index.php/OpenFlow_v1.2, última vez visitada en Agosto '14.

[9] Openflow v1.3. http://archive.openflow.org/wk/index.php/OpenFlow_v1.3, última vez visitada en Agosto '14.

[10] Openflow/SDN: Proven customer benefits. http://www.nec.com/en/global/rd/research/cl/sdn/innovation/page03.html, última vez visitada en Septiembre '14.

[11] Página personal de Shmuel Zacks. http://www.cs.technion.ac.il/people/zaks/, última vez visitada en Noviembre '13.

[12] Ryu 3.13 Documentation. http://ryu.readthedocs.org/en/latest/, última vez visitada en Agosto '14.

[13] Ryu Mailing List. https://lists.sourceforge.net/lists/listinfo/ryu-devel, última vez visitada en Septiembre '14.

[14] SDN Stack. http://networkstatic.net/the-northbound-api-2/, última vez visitada en Septiembre '14.

[15] VirtualBox. https://www.virtualbox.org, última vez visitada en Septiembre '13.

[16] Wha is OpenFlow? https://www.sdncentral.com/what-is-openflow/, última vez visitada en Septiembre '14.

[17] Wha is SDN? https://www.sdncentral.com/what-the-definition-of-software-defined-networking última vez visitada en Septiembre '14.

[18] D. Beazly and B.K. Jones. *Python Cookbook*. O'Reilly, third edition, 2013.

[19] A. de la Oliva, H. Ali-Ahmad, E. Bizouarn, C. Cicconetti, and C. Vitale. Initial specification of Connectivity Management concepts and architecture. pages 59–64, 2013.

[20] Open Networking Foundation. OpenFlow Switch Specifiaction 1.3.4. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf, última vez visitada en Septiembre '14.

[21] Michael Salvador. Pluggin into SDN. http://www.belden.com/blog/datacenters/Plugging-Into-SDN.cfm, última vez visitada en Agosto '14, 2014.

Apéndice A

Planificación y presupuesto.

A.1. Introducción

En este capítulo se va a describir en detalle a planificación de este proyecto así como el presupuesto necesario para llevarlo a cabo.

- **Tarea A: Documentación previa y análisis**
  - **Subtarea A.1: Análisis general de las redes basadas en software**
    - **Descripción:** Se recolectó y estudio documentación general.
    - **Objetivos:** Compresión de conceptos generales del tema a tratar.
    - **Dependencia:** da comienzo a este Trabajo de Fin de Grado.
    - **Duración:** 3 semanas.
  - **Subtarea A.2: Análisis y debug del controlador Ryu**
    - **Descripción:** Se estudió e hizo debug del controlador Ryu.
    - **Objetivos:** Conocer el funcionamiento del controlador.
    - **Dependencia:** Después de la tarea A.1
    - **Duración:** 3 semanas.
  - **Subtarea A.3: Planificación del TFG**
    - **Subtarea A.3.1: Análisis de los bloques del proyecto**
      - **Descripción:** Estudio y análisis de los dos bloques del proyecto.
      - **Objetivos:** Determinar el alcance del proyecto.
      - **Dependencia:** Después de la tarea A.2
      - **Duración:** 2 semanas.
- **Tarea B: Desarrollo del módulo de control de la topología**
  - **Tarea B.1: Análisis y estudio de documentación de Ryu y OF.**
    - **Descripción:** Estudio de eventos relacionados con la topología del controlador Ryu y los mensajes OF.
    - **Objetivos:** Poder centrar la fase de desarrollo en puntos más concretos.
    - **Dependencia:** Después de la tarea A.3.
    - **Duración:** 2 semanas.

- **Tarea B.2: Desarrollo del código de control de topología.**
  - **Subtarea B.2.1: Desarrollo del código python**
    - **Descripción:** Desarrollo del código python correspondiente a este bloque.
    - **Objetivos:** Obtener un módulo capaz de obtener la topología de red.
    - **Dependencia:** Después de la tarea B.1
    - **Duración:** 7 semanas.
  - **Subtarea B.2.2: Pruebas y control de errores**
    - **Descripción:** Pruebas de funcionamiento .
    - **Objetivos:** Comprobar el correcto funcionamiento del módulo.
    - **Dependencia:** Después de la tarea B.2.1
    - **Duración:** 1 semana.

- **Tarea C: Desarrollo de algoritmo de enrutado de nivel 2**
  - **Tarea C.1: Estudio del algoritmo original.**
    - **Descripción:** Estudio del algoritmo original desarrollado por Shmuel Zacks
    - **Objetivos:** Estudiar la manera de proceder para el desarrollo del código python.
    - **Dependencia:** Después de la tarea B.2.
    - **Duración:** 4 semanas.
  - **Tarea C.2: Desarrollo del código del algoritmo.**
    - **Subtarea C.2.1: Desarrollo del código python**
      - **Descripción:** Desarrollo del código python correspondiente a este bloque.
      - **Objetivos:** Obtener un algoritmo capaz de funcionar sobre una red para hacer un estudio de su viabilidad.
      - **Dependencia:** Después de la tarea C.1
      - **Duración:** 10 semana.
    - **Subtarea C.2.2: Pruebas y control de errores**
      - **Descripción:** Pruebas de funcionamiento .
      - **Objetivos:** Comprobar el correcto funcionamiento del módulo.
      - **Dependencia:** Después de la tarea C.2.1
      - **Duración:** 2 semana.

- **Tarea D: Toma de datos**
  - **Tarea D.1: Documentación.**
    - **Descripción:** Se tomaron datos de rendimiento de nuestro algoritmo.
    - **Objetivos:** Comprobar la viabilidad de ejecutar este algoritmo en una red.
    - **Dependencia:** Después de la tarea C.2.2
    - **Duración:** 6 semanas.

- **Tarea E: Memoria.**
  - **Tarea E.1: Redacción de la memoria.**
    - **Descripción:** se redactó esta memoria.
    - **Objetivos:** obtener la memoria del Trabajo de Fin de Grado.
    - **Dependencia:** da comienzo tras la tarea D.1
    - **Duración:** 6 Semanas

Tarea	Duración (semanas)	G. Ing. Tlm. (Ing/m)	Ing. Senior (Ing/m)
Documentación previa y análisis			
A.1 Análisis general de redes basadas en software	3	0.375	-
A.2 Análisis y debug del controlador Ryu	3	0,375	-
A.3 Planificación del TFG	3	0,375	0,25
Total		**1.125**	**0.25**
Desarrollo del módulo de control de la topología			
B.1 Análisis y estudio de documentación de Ryu y OF	2	0.25	-
B.2 Desarrollo del código de control de topología	8	1	-
Total		**1.25**	-
Desarrollo del algoritmo de enrutado de nivel 2			
C.1 Estudio del algoritmo original	4	0.5	0.5
C.2 Desarrollo del código del algoritmo	12	1.5	0.25
Total		**2**	**0.75**
Toma de datos			
D.1 Documentación de la toma de datos	6	0.75	0.5
Total		**0.75**	**0.5**
Memoria			
E.1 Redacción de la memoria	6	0.75	-
Total		**0.75**	-
Total	47	**4.93**	**1.5**

Tabla A.1: Resumen descomposición en tareas

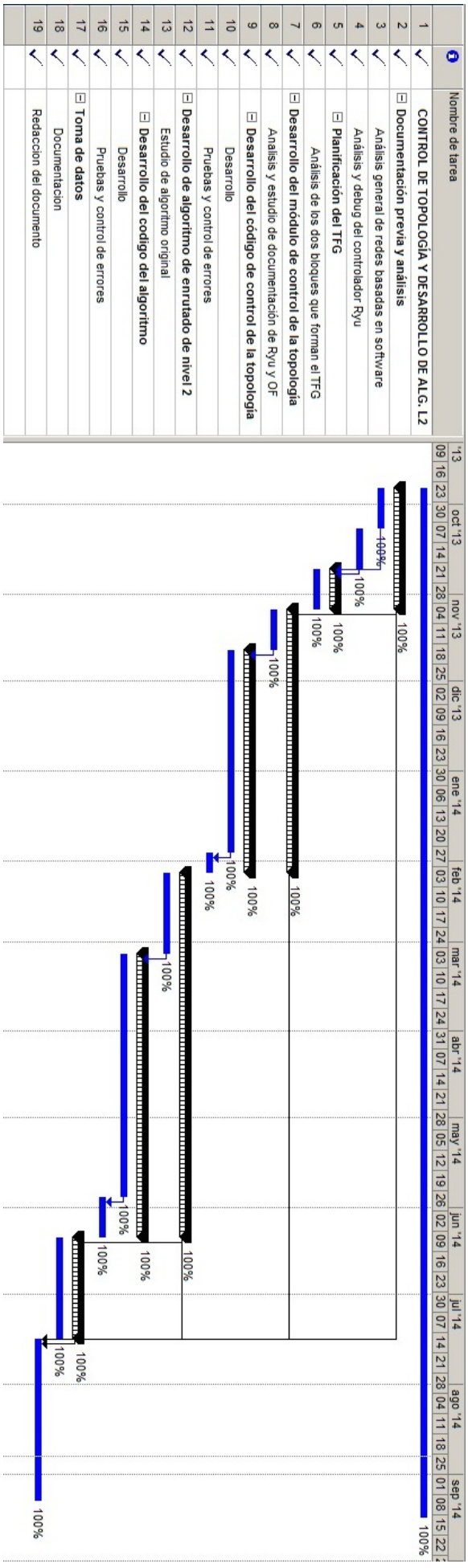

Figura A.1: Diagrama de Gantt del proyecto

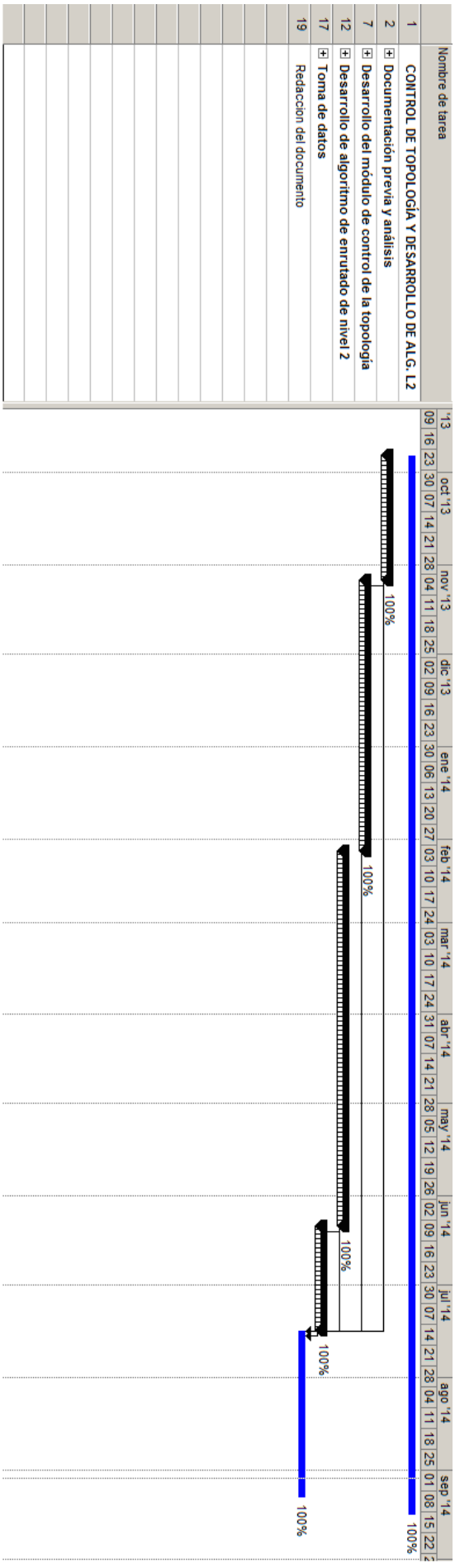

Figura A.2: Diagrama de Gantt resumido del proyecto

A.2. Recursos

A continuación se van a distinguir los diferentes recursos que han sido necesarios para la realización del proyecto.

- **Recursos Materiales**
  - 1 PC Portátil MacBook Pro Intel Core i5 2.3Ghz, 8 Gb RAM con sistema operativo OS X 10.9.4
- **Mano de Obra directa**
  - 1 Graduado en Ingeniería Telemática: 4.93 ingenieros/mes
  - 1 Ingeniero Senior: 1.5 ingenieros/mes

A.3. Presupuesto

1. Autor: Julián Merino Fresno
2. Departamento: Ingeniería Telemática
3. Descripción del Proyecto:
   - Título: Análisis del control de topología en OpenFlow e implementación de algoritmo de enrutado a nivel 2
   - Duración: 8 meses
   - Tasa de costes indirectos: 25 %.
4. Presupuesto total del Proyecto (valorado en Euros): euros. Ver tabla A.2
5. Subcontratación de tareas: no se especifican.
6. Otros costes directos del proyecto: no se especifican.

Concepto	**Cantidad (€)**	**Coste €**	**% Proyecto**	**Dedicación (meses)**	**Depreciación (meses)**	**Total €**
Recursos materiales						
Ordenadores portátiles	1	1250	100	10	60	191,67
Total						**191,67**
Mano de obra directa						
Graduado en Ing. Telemática	1 (4.93 ing/mes)	2.694,39	-	-	-	13.283,34
Ingenieros Senior	2 (1.5 ing/mes)	4.289,54	-	-	-	6.434,31
Total						**19.717,65**
Total de costes directos						**19.909,32 €**
Costes indirectos						
Costes indirectos	-	-	-	-	-	25 %
Total						**4.977,33**
Total						**24.886,65 €**

Tabla A.2: Tabla de presupuesto

Apéndice B

Código del módulo de control de la topología de red

B.1. Introducción

En este apéndice se va a mostrar el código completo del módulo de topología de red.

```
import StringIO
import urllib
import base64
from webob import Response

from ryu.topology import switches, event
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller import dpset
from ryu.controller import mac_to_port
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_0
from ryu.app.wsgi import ControllerBase, WSGIApplication
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.mac import haddr_to_bin

import networkx as nx
import matplotlib.pyplot as plt


class GraphExplorer(app_manager.RyuApp):

    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]
    #The contexts are necessary for some events and resources.
    _CONTEXTS = {'dpset': dpset.DPSet,  # used for EventDP
                 'switches': switches.Switches,  # used for EventLinkAdd
```

```
                'wsgi': WSGIApplication}  # used for REST stuff

    def __init__(self, *args, **kwargs):
        super(GraphExplorer, self).__init__(*args, **kwargs)
        #dictionary used to remember which mac is in which port, based on simple_sw
        self.mac_to_port = {}
        #the topology
        self.topo = nx.Graph()


    def add_flow(self, datapath, in_port, dst, actions):
            """Add a flow to the datapath.
        datapath -- the datapath in which the flow will be installed
        in_port -- the port that reported the packet_in
        dst -- the mac to which the packet was sent
        actions -- the actions (mostly OFPActionOutput)
        Basicly, the datapath received a packet from in_port to be sent
        to dst and wants to do actions to it."""
        ofproto = datapath.ofproto
        #the actions are applied to the packets that match this "match"
        match = datapath.ofproto_parser.OFPMatch(in_port=in_port,
                                                 dl_dst=dst)
        #the flowmod message to install the rule on the datapath
        mod = datapath.ofproto_parser.OFPFlowMod(
              datapath=datapath, match=match, cookie=0,
              command=ofproto.OFPFC_ADD, idle_timeout=0, hard_timeout=0,
              priority=ofproto.OFP_DEFAULT_PRIORITY,
              flags=ofproto.OFPFF_SEND_FLOW_REM, actions=actions)
        datapath.send_msg(mod)
    #@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        """Handle packet_in event, installing the necessary flows.
        ev -- the event and all of it's fields.
        """
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)

        dst = eth.dst
        src = eth.src

        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        self.logger.info("packet in %s %s %s %s", dpid, src, dst, msg.in_port)

        # learn a mac address to avoid FLOOD next time.
```

```
        self.mac_to_port[dpid][src] = msg.in_port

        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            self.add_flow(datapath, msg.in_port, dst, actions)

        out = datapath.ofproto_parser.OFPPacketOut(
            datapath=datapath, buffer_id=msg.buffer_id, in_port=msg.in_port,
            actions=actions)
        datapath.send_msg(out)

    @set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
    def _port_status_handler(self, ev):
        """Report port changed, if port was deleted, remove from topology..
        ev -- PortStatus event, with all of it's fields.
        """
        msg = ev.msg
        reason = msg.reason
        port_no = msg.desc.port_no

        #msg.datapath.id contains the dpid where the port was added
        #msg.desc.port_no contains which port
        ofproto = msg.datapath.ofproto
        if reason == ofproto.OFPPR_ADD:
            self.logger.info("port added %s", port_no)
        elif reason == ofproto.OFPPR_DELETE:

            self.logger.info("port deleted %s", port_no)
        elif reason == ofproto.OFPPR_MODIFY:
            self.logger.info("port modified %s", port_no)
        else:
            self.logger.info("Illeagal port state %s %s", port_no, reason)

    @set_ev_cls(dpset.EventDP, MAIN_DISPATCHER)
    def dp_handler(self, ev):
        """Update topology graph when a switch enters or leaves.
        ev -- datapath event and all of it's fields

        Datos:
        ev.dp.id: dpid que genera el evento
        ev.enter es true si el dp esta entrando
        """
        if ev.dp.id is None:
```

```
            return

        if ev.enter:
            self.topo.add_node(ev.dp.id)
            self.logger.info('Switch %s anadido a la topologia', str(ev.dp.id))
        else:
            self.topo.remove_node(ev.dp.id)
            self.logger.info('Switch %s borrado de la topologia',
                             str(ev.dp.id))



    @set_ev_cls(event.EventLinkAdd, MAIN_DISPATCHER)
    def link_handler(self, ev):
        """Add new links to the topology graph
        ev -- LinkAdd event and all of it's fields.

        Datos:
        ev.link.src: pareja de datos (srcdpid, srcport)
        ev.link.dst: pareja de datos (dstdpid, dstport)"""

        src = ev.link.src
        dst = ev.link.dst

        if src.dpid is None or dst.dpid is None:
            return
        self.topo.add_edge(src.dpid, dst.dpid)
        self.logger.info('Edge, la topologia hasta el momento es: %s',
                         str(self.topo.edges()))
```

Apéndice C

Código del algoritmo de enrutado de nivel 2

C.1. Introducción

En este apéndice se va a mostrar el código completo del algoritmo de enrutado. Las decisiones de diseño se han especificado anteriormente en este mismo documento.

```
import networkx as nx
import random
import sys

number_of_nodes = 0
s = 0
h = 0
b = 0

number_of_nodes = int(raw_input("Introduce el numero de nodos (25/50/100/125/150)   "))

s = int(raw_input("Introduce el nodo origen   "))
if s > number_of_nodes-1:
        print "Nodo origen NO VALIDO"
        sys.exit()
h = int(raw_input("Introduce el nodo destino del path a seguir  "))
if h > number_of_nodes-1 or h == s:
        print "Nodo destino del path a seguir NO VALIDO"
        sys.exit()
b = int(raw_input("Introduce el nodo destino   "))
if b > number_of_nodes-1 or b == s or b == h:
        print "Nodo destino NO VALIDO"
        sys.exit()
i=0
G = nx.gnp_random_graph(number_of_nodes, 0.0725)
k = 0

distancia =dict( G.nodes(data=True));
pesos = dict(G.nodes(data=True));
try:
        P = nx.dijkstra_path(G,s,h)
except nx.NetworkXNoPath:
        print "No existe ningun camino entre ",s," y ",h
        sys.exit()
dist_original = nx.shortest_path_length(G,s,h)
try:
        dist_b = nx.shortest_path_length(G,s,b)
except nx.NetworkXNoPath:
        print "No existe ningun camino entre ",s," y ",b
        sys.exit()
```

```
if dist_b > dist_original:
        print "La distancia a B es mayor que la distancia del Path Original (...)"
        sys.exit()


for key in distancia:
        distancia[key]=-1
for key in pesos:
        pesos[key]=0

distancia[s] = 0
pesos[s] = 1

while distancia[b] == -1:
        current = nx.single_source_shortest_path_length(G,s,i)
        siguientes = nx.single_source_shortest_path_length(G,s,i+1)
        print current

        next_nodes = []
        current_nodes = []

        for key in siguientes.keys():
                if siguientes[key] == i+1:
                        next_nodes.append(key)

        for key in current.keys():
                if current[key]== i:
                        current_nodes.append(key)

        for k in range(len(next_nodes)):
                distancia[next_nodes[k]]=i+1

        for k in range(len(next_nodes)):
                for v in range(len(current_nodes)):
                        if G.has_edge(next_nodes[k],current_nodes[v]):
                                if next_nodes[k] in P:
                                    pesos[next_nodes[k]] = max(pesos[next_nodes[k]],pesos[current_nodes[v]]+1)
                                else:
                                    pesos[next_nodes[k]] = max(pesos[next_nodes[k]],pesos[current_nodes[v]])

        i = i+1
Path = []
Path.append(b)
anterior = b
mayor = b
while i>0:

        current = nx.single_source_shortest_path_length(G,s,i-1)

        siguientes_nodos=[]

        for key in current.keys():
                if current[key]==i-1:
                        siguientes_nodos.append(key)

        print siguientes_nodos
        aux = 0
        for k in range(len(siguientes_nodos)):
                if G.has_edge(anterior,siguientes_nodos[k]):
                        if pesos[siguientes_nodos[k]]>aux:
                                aux = pesos[siguientes_nodos[k]]
                                mayor = siguientes_nodos[k]
        anterior = mayor
        Path.append(mayor)
        i=i-1
tam = len(Path)
print tam
Path_aux = []
while tam > 0:
        tam = tam -1
        Path_aux.append(Path[tam])
```

```
print "Numero de nodos"
print G.number_of_nodes()
print "Numero de Edges"
print G.number_of_edges()
print "El path Original de Dijkstra"
print P
print "Path con el alg de Shmuel"
print Path_aux
print "Path de dijkstra"
print nx.dijkstra_path(G,s,b)
```

Apéndice D

Datos recogidos

D.1. Introducción

En este apéndice se van a mostrar todos los datos recogidos sobre los cuales se basa la conclusión de este proyecto. Se mostrarán los datos para redes de 150, 200, 250 y 300 nodos, cada unas de ellas con 4 probabilidades de enlace, como se ha explicado con anterioridad.

Las columnas de las tablas de datos tienen el siguiente formato:

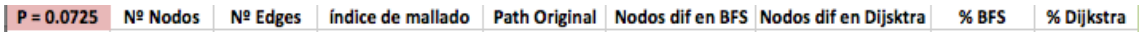

Figura D.1: Columnas de las tablas de datos recogidos

- **Nº Nodos**: Número de nodos de la red sobre la cual se recoge la muestra.
- **Nº de Edges**: Número de enlaces en toda la red.
- **Indice de mallado**: Índice que indica cuantos enlaces, en media, tiene cada nodo.
- **Path Original**: Número de saltos del path original.
- **Nodos Dif Dijkstra**: Número nodos diferentes al path original del nuevo path calculado con Dijkstra.
- **Nodos Dif BFS**: Número nodos diferentes al path original del nuevo path calculado con el algoritmo desarrollado.
- **% BFS**: Porcentaje de nodos iguales al path original del path de calculado por el algoritmo desarrollado.
- **% Dijkstra**: Porcentaje de nodos iguales al path original del path de calculado por el algoritmo de Dijkstra.

En todas las tablas encontraremos un dato llamado *DIFF* que indica la diferencia en media del porcentaje de nodos iguales al path original del algoritmo desarrollado con el algoritmo de Dijkstra.

A continuación se mostrarán todas las tablas:

Para las redes de **150** nodos:

P = 0.025	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	150	309	2,060	4	2	2	50,00%	50,00%
Muestra 2	150	282	1,880	4	3	3	75,00%	75,00%
Muestra 3	150	277	1,847	5	2	2	40,00%	40,00%
Muestra 4	150	268	1,787	5	1	1	20,00%	20,00%
Muestra 5	150	281	1,873	6	3	3	50,00%	50,00%
Muestra 6	150	305	2,033	5	4	4	80,00%	80,00%
Muestra 7	150	275	1,833	6	2	2	33,33%	33,33%
Muestra 8	150	287	1,913	4	1	1	25,00%	25,00%
Muestra 9	150	299	1,993	6	2	2	33,33%	33,33%
Muestra 10	150	287	1,913	7	1	3	14,29%	42,86%
Muestra 11	150	263	1,753	4	2	2	50,00%	50,00%
Muestra 12	150	286	1,907	6	3	4	50,00%	66,67%
Muestra 13	150	276	1,840	4	3	3	75,00%	75,00%
Muestra 14	150	307	2,047	5	3	3	60,00%	60,00%
Muestra 15	150	294	1,960	4	2	2	50,00%	50,00%
Muestra 16	150	280	1,867	6	4	4	66,67%	66,67%
Muestra 17	150	311	2,073	5	2	2	40,00%	40,00%
Muestra 18	150	271	1,807	5	2	2	40,00%	40,00%
Muestra 19	150	270	1,800	4	0	0	0,00%	0,00%
Muestra 20	150	258	1,720	5	3	4	60,00%	80,00%
Muestra 21	150	278	1,853	5	3	3	60,00%	60,00%
Muestra 22	150	261	1,740	4	1	1	25,00%	25,00%
Muestra 23	150	293	1,953	6	3	3	50,00%	50,00%
Muestra 24	150	291	1,940	5	3	3	60,00%	60,00%
Muestra 25	150	269	1,793	5	2	2	40,00%	40,00%
Muestra 26	150	297	1,980	6	5	5	83,33%	83,33%
Muestra 27	150	287	1,913	4	3	3	75,00%	75,00%
Muestra 28	150	293	1,953	5	2	2	40,00%	40,00%
Muestra 29	150	328	2,187	6	1	1	16,67%	16,67%
Muestra 30	150	282	1,880	5	3	3	60,00%	60,00%
	150	285,5	1,903	5,033333333	2,366666667	2,5	47,42%	49,60%
							DIFF	2,17%

Figura D.2: Datos para una red de 150 nodos con una probabilidad de enlace de 0.025

P = 0.0375	Nº Nodos	Nº Edges	dice de malla	Path Original	odos dif en B	os dif en Dijs	% BFS	% Dijkstra
Muestra 1	150	445	2,97	5	3	3	60,00%	60,00%
Muestra 2	150	414	2,76	4	3	3	75,00%	75,00%
Muestra 3	150	396	2,64	6	3	5	50,00%	83,33%
Muestra 4	150	427	2,85	3	2	2	66,67%	66,67%
Muestra 5	150	455	3,03	4	2	2	50,00%	50,00%
Muestra 6	150	422	2,81	4	3	3	75,00%	75,00%
Muestra 7	150	418	2,79	4	3	3	75,00%	75,00%
Muestra 8	150	426	2,84	4	2	2	50,00%	50,00%
Muestra 9	150	377	2,51	5	3	3	60,00%	60,00%
Muestra 10	150	443	2,95	5	3	4	60,00%	80,00%
Muestra 11	150	392	2,61	4	0	0	0,00%	0,00%
Muestra 12	150	395	2,63	5	1	1	20,00%	20,00%
Muestra 13	150	468	3,12	4	2	2	50,00%	50,00%
Muestra 14	150	398	2,65	5	1	1	20,00%	20,00%
Muestra 15	150	412	2,75	5	3	4	60,00%	80,00%
Muestra 16	150	416	2,77	4	1	1	25,00%	25,00%
Muestra 17	150	431	2,87	4	1	1	25,00%	25,00%
Muestra 18	150	421	2,81	5	4	4	80,00%	80,00%
Muestra 19	150	397	2,65	4	2	2	50,00%	50,00%
Muestra 20	150	400	2,67	4	2	2	50,00%	50,00%
Muestra 21	150	431	2,87	4	3	3	75,00%	75,00%
Muestra 22	150	401	2,67	4	3	3	75,00%	75,00%
Muestra 23	150	448	2,99	4	2	2	50,00%	50,00%
Muestra 24	150	409	2,73	4	3	3	75,00%	75,00%
Muestra 25	150	405	2,70	4	3	3	75,00%	75,00%
Muestra 26	150	424	2,83	4	3	3	75,00%	75,00%
Muestra 27	150	406	2,71	4	3	3	75,00%	75,00%
Muestra 28	150	408	2,72	4	3	3	75,00%	75,00%
Muestra 29	150	409	2,73	5	4	4	80,00%	80,00%
Muestra 30	150	440	2,93	5	2	2	40,00%	40,00%
			2,78533333	4,33333333			56,56%	59,00%
							DIFF	2,44%

Figura D.3: Datos para una red de 150 nodos con una probabilidad de enlace de 0.0375

P = 0.05	Nº Nodos	Nº Edges	dice de malla	Path Original	bdos dif en B	os dif en Dijs	% BFS	% Dijkstra
Muestra 1	150	537	3,58	4	3	3	75,00%	75,00%
Muestra 2	150	542	3,61	5	2	3	40,00%	60,00%
Muestra 3	150	556	3,71	4	1	1	25,00%	25,00%
Muestra 4	150	555	3,70	4	2	3	50,00%	75,00%
Muestra 5	150	582	3,88	4	2	2	50,00%	50,00%
Muestra 6	150	542	3,61	4	1	1	25,00%	25,00%
Muestra 7	150	543	3,62	4	3	3	75,00%	75,00%
Muestra 8	150	550	3,67	4	2	3	50,00%	75,00%
Muestra 9	150	548	3,65	4	1	1	25,00%	25,00%
Muestra 10	150	582	3,88	4	2	2	50,00%	50,00%
Muestra 11	150	564	3,76	5	3	3	60,00%	60,00%
Muestra 12	150	585	3,90	4	3	3	75,00%	75,00%
Muestra 13	150	547	3,65	5	3	4	60,00%	80,00%
Muestra 14	150	539	3,59	4	2	2	50,00%	50,00%
Muestra 15	150	537	3,58	4	1	1	25,00%	25,00%
Muestra 16	150	539	3,59	4	1	1	25,00%	25,00%
Muestra 17	150	572	3,81	4	3	3	75,00%	75,00%
Muestra 18	150	602	4,01	4	3	3	75,00%	75,00%
Muestra 19	150	546	3,64	5	3	3	60,00%	60,00%
Muestra 20	150	541	3,61	3	1	1	33,33%	33,33%
Muestra 21	150	597	3,98	3	1	1	33,33%	33,33%
Muestra 22	150	587	3,91	4	2	2	50,00%	50,00%
Muestra 23	150	570	3,80	4	2	2	50,00%	50,00%
Muestra 24	150	556	3,71	3	2	2	66,67%	66,67%
Muestra 25	150	559	3,73	4	2	2	50,00%	50,00%
Muestra 26	150	565	3,77	3	1	1	33,33%	33,33%
Muestra 27	150	538	3,59	4	3	3	75,00%	75,00%
Muestra 28	150	484	3,23	4	2	2	50,00%	50,00%
Muestra 29	150	545	3,63	4	3	3	75,00%	75,00%
Muestra 30	150	528	3,52	4	3	3	75,00%	75,00%
			3,69733333	**4**			**52,06%**	**55,06%**
							DIFF	**3,00%**

Figura D.4: Datos para una red de 150 nodos con una probabilidad de enlace de 0.05

P = 0.0725	Nº Nodos	Nº Edges	dice de malla	Path Original	bdos dif en B	os dif en Dijs	% BFS	% Dijkstra
Muestra 1	150	802	5,35	3	2	2	66,67%	66,67%
Muestra 2	150	783	5,22	4	3	3	75,00%	75,00%
Muestra 3	150	830	5,53	3	1	1	33,33%	33,33%
Muestra 4	150	777	5,18	3	2	2	66,67%	66,67%
Muestra 5	150	776	5,17	4	2	3	50,00%	75,00%
Muestra 6	150	783	5,22	3	2	2	66,67%	66,67%
Muestra 7	150	849	5,66	4	3	3	75,00%	75,00%
Muestra 8	150	800	5,33	3	2	2	66,67%	66,67%
Muestra 9	150	842	5,61	4	2	2	50,00%	50,00%
Muestra 10	150	825	5,50	4	1	2	25,00%	50,00%
Muestra 11	150	798	5,32	4	1	1	25,00%	25,00%
Muestra 12	150	840	5,60	4	3	3	75,00%	75,00%
Muestra 13	150	786	5,24	4	3	3	75,00%	75,00%
Muestra 14	150	849	5,66	4	2	3	50,00%	75,00%
Muestra 15	150	812	5,41	3	2	2	66,67%	66,67%
Muestra 16	150	781	5,21	3	2	2	66,67%	66,67%
Muestra 17	150	853	5,69	4	1	1	25,00%	25,00%
Muestra 18	150	801	5,34	4	2	3	50,00%	75,00%
Muestra 19	150	839	5,59	4	2	2	50,00%	50,00%
Muestra 20	150	817	5,45	4	2	2	50,00%	50,00%
Muestra 21	150	827	5,51	4	2	3	50,00%	75,00%
Muestra 22	150	798	5,32	4	3	3	75,00%	75,00%
Muestra 23	150	804	5,36	4	0	0	0,00%	0,00%
Muestra 24	150	789	5,26	4	1	1	25,00%	25,00%
Muestra 25	150	768	5,12	4	2	2	50,00%	50,00%
Muestra 26	150	821	5,47	4	3	3	75,00%	75,00%
Muestra 27	150	823	5,49	4	2	2	50,00%	50,00%
Muestra 28	150	817	5,45	4	2	3	50,00%	75,00%
Muestra 29	150	821	5,47	4	2	3	50,00%	75,00%
Muestra 30	150	808	5,39	4	3	3	75,00%	75,00%
			5,40422222	**3,76666667**			**53,61%**	**59,44%**
							DIFF	**5,83%**

Figura D.5: Datos para una red de 150 nodos con una probabilidad de enlace de 0.0725

P = 0.1	Nº Nodos	Nº Edges	dice de malla	Path Original	odos dif en B	os dif en Dijs	% BFS	% Dijkstra
Muestra 1	150	1151	7,673	4	1	1	25,00%	25,00%
Muestra 2	150	1097	7,313	3	2	2	66,67%	66,67%
Muestra 3	150	1165	7,767	3	2	2	66,67%	66,67%
Muestra 4	150	1090	7,267	4	2	2	50,00%	50,00%
Muestra 5	150	1159	7,727	3	2	2	66,67%	66,67%
Muestra 6	150	1108	7,387	3	2	2	66,67%	66,67%
Muestra 7	150	1051	7,007	3	2	2	66,67%	66,67%
Muestra 8	150	1106	7,373	3	2	2	66,67%	66,67%
Muestra 9	150	1071	7,140	4	2	3	50,00%	75,00%
Muestra 10	150	1137	7,580	3	2	2	66,67%	66,67%
Muestra 11	150	1141	7,607	3	2	2	66,67%	66,67%
Muestra 12	150	1103	7,353	3	2	2	66,67%	66,67%
Muestra 13	150	1118	7,453	3	2	2	66,67%	66,67%
Muestra 14	150	1066	7,107	4	2	2	50,00%	50,00%
Muestra 15	150	1120	7,467	3	2	2	66,67%	66,67%
Muestra 16	150	1136	7,573	3	2	2	66,67%	66,67%
Muestra 17	150	1111	7,407	3	1	1	33,33%	33,33%
Muestra 18	150	1117	7,447	4	2	2	50,00%	50,00%
Muestra 19	150	1087	7,247	3	1	1	33,33%	33,33%
Muestra 20	150	1120	7,467	4	2	2	50,00%	50,00%
Muestra 21	150	1135	7,567	3	2	2	66,67%	66,67%
Muestra 22	150	1122	7,480	3	2	2	66,67%	66,67%
Muestra 23	150	1144	7,627	3	2	2	66,67%	66,67%
Muestra 24	150	1141	7,607	3	2	2	66,67%	66,67%
Muestra 25	150	1113	7,420	3	1	1	33,33%	33,33%
Muestra 26	150	1082	7,213	3	2	2	66,67%	66,67%
Muestra 27	150	1112	7,413	3	2	2	66,67%	66,67%
Muestra 28	150	1084	7,227	3	2	2	66,67%	66,67%
Muestra 29	150	1117	7,447	4	1	1	25,00%	25,00%
Muestra 30	150	1101	7,340	3	1	1	33,33%	33,33%
			7,423	**3,23333333**			**56,67%**	**57,50%**
							DIFF	**0,83%**

Figura D.6: Datos para una red de 150 nodos con una probabilidad de enlace de 0.1

Para las redes de **200** nodos:

P = 0.025	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	200	469	2,345	3	2	2	66,67%	66,67%
Muestra 2	200	493	2,465	5	3	3	60,00%	60,00%
Muestra 3	200	525	2,625	4	3	3	75,00%	75,00%
Muestra 4	200	497	2,485	6	3	3	50,00%	50,00%
Muestra 5	200	491	2,455	4	4	4	100,00%	100,00%
Muestra 6	200	498	2,490	5	3	4	60,00%	80,00%
Muestra 7	200	471	2,355	6	4	4	66,67%	66,67%
Muestra 8	200	518	2,590	5	1	1	20,00%	20,00%
Muestra 9	200	526	2,630	5	3	4	60,00%	80,00%
Muestra 10	200	485	2,425	5	4	4	80,00%	80,00%
Muestra 11	200	487	2,435	5	3	3	60,00%	60,00%
Muestra 12	200	502	2,510	4	3	3	75,00%	75,00%
Muestra 13	200	454	2,270	4	3	3	75,00%	75,00%
Muestra 14	200	500	2,500	4	1	1	25,00%	25,00%
Muestra 15	200	487	2,435	4	2	2	50,00%	50,00%
Muestra 16	200	473	2,365	5	2	2	40,00%	40,00%
Muestra 17	200	433	2,165	5	3	2	60,00%	40,00%
Muestra 18	200	478	2,390	4	2	2	50,00%	50,00%
Muestra 19	200	526	2,630	5	2	2	40,00%	40,00%
Muestra 20	200	466	2,330	4	2	2	50,00%	50,00%
Muestra 21	200	487	2,435	5	4	4	80,00%	80,00%
Muestra 22	200	483	2,415	4	2	2	50,00%	50,00%
Muestra 23	200	496	2,480	7	2	5	28,57%	71,43%
Muestra 24	200	506	2,530	5	2	3	40,00%	60,00%
Muestra 25	200	483	2,415	6	3	3	50,00%	50,00%
Muestra 26	200	501	2,505	6	3	3	50,00%	50,00%
Muestra 27	200	451	2,255	5	4	4	80,00%	80,00%
Muestra 28	200	473	2,365	5	3	3	60,00%	60,00%
Muestra 29	200	491	2,455	4	1	1	25,00%	25,00%
Muestra 30	200	505	2,525	5	3	4	60,00%	80,00%
	200	**488,5**	**2,443**	**4,8**	**2,666666667**	**2,866666667**	**56,23%**	**59,66%**
							DIFF	**3,43%**

Figura D.7: Datos para una red de 200 nodos con una probabilidad de enlace de 0.025

P = 0.0375	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	200	735	3,675	4	3	3	75,00%	75,00%
Muestra 2	200	745	3,725	4	1	1	25,00%	25,00%
Muestra 3	200	718	3,590	4	2	2	50,00%	50,00%
Muestra 4	200	729	3,645	4	3	3	75,00%	75,00%
Muestra 5	200	788	3,940	4	3	3	75,00%	75,00%
Muestra 6	200	708	3,540	4	2	2	50,00%	50,00%
Muestra 7	200	737	3,685	4	2	2	50,00%	50,00%
Muestra 8	200	694	3,470	5	3	4	60,00%	80,00%
Muestra 9	200	694	3,470	4	1	1	25,00%	25,00%
Muestra 10	200	735	3,675	4	3	3	75,00%	75,00%
Muestra 11	200	747	3,735	5	3	3	60,00%	60,00%
Muestra 12	200	806	4,030	4	2	3	50,00%	75,00%
Muestra 13	200	796	3,980	4	1	1	25,00%	25,00%
Muestra 14	200	757	3,785	4	1	1	25,00%	25,00%
Muestra 15	200	755	3,775	3	2	2	66,67%	66,67%
Muestra 16	200	720	3,600	5	0	0	0,00%	0,00%
Muestra 17	200	786	3,930	4	2	2	50,00%	50,00%
Muestra 18	200	769	3,845	3	2	2	66,67%	66,67%
Muestra 19	200	784	3,920	4	2	2	50,00%	50,00%
Muestra 20	200	707	3,535	4	3	3	75,00%	75,00%
Muestra 21	200	774	3,870	4	3	3	75,00%	75,00%
Muestra 22	200	765	3,825	3	2	2	66,67%	66,67%
Muestra 23	200	765	3,825	5	3	3	60,00%	60,00%
Muestra 24	200	713	3,565	4	3	3	75,00%	75,00%
Muestra 25	200	724	3,620	4	3	3	75,00%	75,00%
Muestra 26	200	746	3,730	4	3	3	75,00%	75,00%
Muestra 27	200	751	3,755	4	1	1	25,00%	25,00%
Muestra 28	200	747	3,735	4	3	3	75,00%	75,00%
Muestra 29	200	747	3,735	5	3	3	60,00%	60,00%
Muestra 30	200	725	3,625	4	3	3	75,00%	75,00%
	200	745,566667	3,728	4,066666667	2,266666667	2,333333333	56,33%	57,83%
							DIFF	1,50%

Figura D.8: Datos para una red de 200 nodos con una probabilidad de enlace de 0.0375

P = 0.05	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	200	993	4,97	3	2	2	66,67%	66,67%
Muestra 2	200	985	4,93	4	2	2	50,00%	50,00%
Muestra 3	200	926	4,63	4	2	2	50,00%	50,00%
Muestra 4	200	1014	5,07	4	2	2	50,00%	50,00%
Muestra 5	200	980	4,90	4	2	2	50,00%	50,00%
Muestra 6	200	984	4,92	4	3	3	75,00%	75,00%
Muestra 7	200	1031	5,16	4	3	3	75,00%	75,00%
Muestra 8	200	1002	5,01	3	2	2	66,67%	66,67%
Muestra 9	200	998	4,99	4	1	1	25,00%	25,00%
Muestra 10	200	1009	5,05	3	2	2	66,67%	66,67%
Muestra 11	200	994	4,97	4	3	3	75,00%	75,00%
Muestra 12	200	968	4,84	4	2	2	50,00%	50,00%
Muestra 13	200	1023	5,12	4	2	3	50,00%	75,00%
Muestra 14	200	1010	5,05	3	2	2	66,67%	66,67%
Muestra 15	200	1017	5,09	4	2	2	50,00%	50,00%
Muestra 16	200	1026	5,13	3	1	1	33,33%	33,33%
Muestra 17	200	1082	5,41	4	2	2	50,00%	50,00%
Muestra 18	200	982	4,91	4	1	1	25,00%	25,00%
Muestra 19	200	972	4,86	4	2	2	50,00%	50,00%
Muestra 20	200	1017	5,09	3	2	2	66,67%	66,67%
Muestra 21	200	1002	5,01	4	2	3	50,00%	75,00%
Muestra 22	200	1028	5,14	4	3	3	75,00%	75,00%
Muestra 23	200	1000	5,00	4	2	2	50,00%	50,00%
Muestra 24	200	1023	5,12	4	2	2	50,00%	50,00%
Muestra 25	200	1005	5,03	4	2	2	50,00%	50,00%
Muestra 26	200	1033	5,17	3	2	2	66,67%	66,67%
Muestra 27	200	968	4,84	4	2	2	50,00%	50,00%
Muestra 28	200	986	4,93	4	2	3	50,00%	75,00%
Muestra 29	200	1017	5,09	4	2	2	50,00%	50,00%
Muestra 30	200	1042	5,21	4	2	2	50,00%	50,00%
			5,0195	3,766666667			54,44%	56,94%
							DIFF	2,50%

Figura D.9: Datos para una red de 200 nodos con una probabilidad de enlace de 0.05

P = 0.0725	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	200	1462	7,310	3	2	2	66,67%	66,67%
Muestra 2	200	1461	7,305	4	1	1	25,00%	25,00%
Muestra 3	200	1459	7,295	4	2	3	50,00%	75,00%
Muestra 4	200	1453	7,265	4	2	2	50,00%	50,00%
Muestra 5	200	1516	7,580	4	2	2	50,00%	50,00%
Muestra 6	200	1484	7,420	3	2	2	66,67%	66,67%
Muestra 7	200	1449	7,245	3	2	2	66,67%	66,67%
Muestra 8	200	1397	6,985	4	1	2	25,00%	50,00%
Muestra 9	200	1435	7,175	3	2	2	66,67%	66,67%
Muestra 10	200	1492	7,460	3	2	2	66,67%	66,67%
Muestra 11	200	1426	7,130	3	1	1	33,33%	33,33%
Muestra 12	200	1425	7,125	4	2	2	50,00%	50,00%
Muestra 13	200	1448	7,240	3	2	2	66,67%	66,67%
Muestra 14	200	1351	6,755	3	2	2	66,67%	66,67%
Muestra 15	200	1442	7,210	4	2	2	50,00%	50,00%
Muestra 16	200	1392	6,960	4	2	2	50,00%	50,00%
Muestra 17	200	1440	7,200	3	2	2	66,67%	66,67%
Muestra 18	200	1471	7,355	3	2	2	66,67%	66,67%
Muestra 19	200	1386	6,930	3	2	2	66,67%	66,67%
Muestra 20	200	1438	7,190	3	1	1	33,33%	33,33%
Muestra 21	200	1470	7,350	4	1	1	25,00%	25,00%
Muestra 22	200	1433	7,165	3	1	1	33,33%	33,33%
Muestra 23	200	1460	7,300	3	1	1	33,33%	33,33%
Muestra 24	200	1446	7,230	3	2	2	66,67%	66,67%
Muestra 25	200	1476	7,380	3	2	2	66,67%	66,67%
Muestra 26	200	1464	7,320	4	2	2	50,00%	50,00%
Muestra 27	200	1456	7,280	3	2	2	66,67%	66,67%
Muestra 28	200	1451	7,255	4	2	2	50,00%	50,00%
Muestra 29	200	1500	7,500	3	2	2	66,67%	66,67%
Muestra 30	200	1380	6,900	3	1	1	33,33%	33,33%
	200	**1445,43333**	**7,227**	**3,366666667**	**1,733333333**	**1,8**	**52,50%**	**54,17%**
							DIFF	**1,67%**

Figura D.10: Datos para una red de 200 nodos con una probabilidad de enlace de 0.0725

P = 0.1	Nº Nodos	Nº Edges	dice de malla	Path Original	odos dif en B	os dif en Dijs	% BFS	% Dijkstra
Muestra 1	200	1915	9,575	3	2	2	66,67%	66,67%
Muestra 2	200	2041	10,205	3	2	2	66,67%	66,67%
Muestra 3	200	1944	9,720	3	2	2	66,67%	66,67%
Muestra 4	200	1966	9,830	3	2	2	66,67%	66,67%
Muestra 5	200	2075	10,375	3	2	2	66,67%	66,67%
Muestra 6	200	2010	10,050	3	2	2	66,67%	66,67%
Muestra 7	200	1993	9,965	3	2	2	66,67%	66,67%
Muestra 8	200	1969	9,845	3	2	2	66,67%	66,67%
Muestra 9	200	2001	10,005	3	2	2	66,67%	66,67%
Muestra 10	200	1969	9,845	3	2	2	66,67%	66,67%
Muestra 11	200	2041	10,205	3	2	2	66,67%	66,67%
Muestra 12	200	2013	10,065	3	1	1	33,33%	33,33%
Muestra 13	200	1937	9,685	3	2	2	66,67%	66,67%
Muestra 14	200	1966	9,830	3	2	2	66,67%	66,67%
Muestra 15	200	2007	10,035	3	2	2	66,67%	66,67%
Muestra 16	200	2049	10,245	3	2	2	66,67%	66,67%
Muestra 17	200	2037	10,185	3	1	1	33,33%	33,33%
Muestra 18	200	1949	9,745	3	1	1	33,33%	33,33%
Muestra 19	200	2009	10,045	3	2	2	66,67%	66,67%
Muestra 20	200	2062	10,310	3	2	2	66,67%	66,67%
Muestra 21	200	2017	10,085	3	2	2	66,67%	66,67%
Muestra 22	200	2016	10,080	4	2	2	50,00%	50,00%
Muestra 23	200	1993	9,965	3	2	2	66,67%	66,67%
Muestra 24	200	2015	10,075	3	1	2	33,33%	66,67%
Muestra 25	200	2013	10,065	3	2	2	66,67%	66,67%
Muestra 26	200	1969	9,845	4	2	2	50,00%	50,00%
Muestra 27	200	2018	10,090	3	2	2	66,67%	66,67%
Muestra 28	200	2001	10,005	3	2	2	66,67%	66,67%
Muestra 29	200	1984	9,920	3	2	2	66,67%	66,67%
Muestra 30	200	1991	9,955	3	2	2	66,67%	66,67%
			9,995	**3,06666667**			**61,11%**	**62,22%**
							DIFF	**1,11%**

Figura D.11: Datos para una red de 200 nodos con una probabilidad de enlace de 0.1

Para las redes de **250** nodos:

P = 0.025	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	250	866	3,464	4	3	3	75,00%	75,00%
Muestra 2	250	800	3,200	6	4	4	66,67%	66,67%
Muestra 3	250	780	3,120	4	2	2	50,00%	50,00%
Muestra 4	250	781	3,124	5	3	3	60,00%	60,00%
Muestra 5	250	808	3,232	4	2	3	50,00%	75,00%
Muestra 6	250	775	3,100	5	4	4	80,00%	80,00%
Muestra 7	250	785	3,140	4	2	2	50,00%	50,00%
Muestra 8	250	771	3,084	4	3	3	75,00%	75,00%
Muestra 9	250	767	3,068	5	3	3	60,00%	60,00%
Muestra 10	250	842	3,368	5	0	0	0,00%	0,00%
Muestra 11	250	839	3,356	4	3	3	75,00%	75,00%
Muestra 12	250	800	3,200	5	4	4	80,00%	80,00%
Muestra 13	250	800	3,200	5	2	2	40,00%	40,00%
Muestra 14	250	783	3,132	5	2	2	40,00%	40,00%
Muestra 15	250	687	2,748	4	2	2	50,00%	50,00%
Muestra 16	250	811	3,244	5	2	2	40,00%	40,00%
Muestra 17	250	768	3,072	5	0	0	0,00%	0,00%
Muestra 18	250	749	2,996	4	3	3	75,00%	75,00%
Muestra 19	250	818	3,272	5	4	4	80,00%	80,00%
Muestra 20	250	774	3,096	4	1	1	25,00%	25,00%
Muestra 21	250	812	3,248	4	3	3	75,00%	75,00%
Muestra 22	250	779	3,116	5	3	4	60,00%	80,00%
Muestra 23	250	786	3,144	4	3	3	75,00%	75,00%
Muestra 24	250	812	3,248	5	3	3	60,00%	60,00%
Muestra 25	250	776	3,104	3	2	2	66,67%	66,67%
Muestra 26	250	789	3,156	5	2	4	40,00%	80,00%
Muestra 27	250	828	3,312	6	3	3	50,00%	50,00%
Muestra 28	250	769	3,076	5	4	4	80,00%	80,00%
Muestra 29	250	781	3,124	5	3	3	60,00%	60,00%
Muestra 30	250	768	3,072	4	2	3	50,00%	75,00%
	250	**790,133333**	**3,161**	**4,6**	**2,566666667**	**2,733333333**	**56,28%**	**59,94%**
							DIFF	**3,67%**

Figura D.12: Datos para una red de 250 nodos con una probabilidad de enlace de 0.025

P = 0.0375	Nº Nodos		índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	250	1201	4,804	4	2	2	50,00%	50,00%
Muestra 2	250	1218	4,872	4	2	2	50,00%	50,00%
Muestra 3	250	1147	4,588	3	2	2	66,67%	66,67%
Muestra 4	250	1141	4,564	4	2	2	50,00%	50,00%
Muestra 5	250	1164	4,656	4	2	2	50,00%	50,00%
Muestra 6	250	1234	4,936	4	3	3	75,00%	75,00%
Muestra 7	250	1160	4,640	4	2	2	50,00%	50,00%
Muestra 8	250	1175	4,700	3	2	2	66,67%	66,67%
Muestra 9	250	1146	4,584	3	0	0	0,00%	0,00%
Muestra 10	250	1155	4,620	5	2	2	40,00%	40,00%
Muestra 11	250	1186	4,744	4	2	2	50,00%	50,00%
Muestra 12	250	1218	4,872	3	2	2	66,67%	66,67%
Muestra 13	250	1220	4,880	5	3	3	60,00%	60,00%
Muestra 14	250	1118	4,472	3	2	2	66,67%	66,67%
Muestra 15	250	1114	4,456	4	3	3	75,00%	75,00%
Muestra 16	250	1134	4,536	3	2	2	66,67%	66,67%
Muestra 17	250	1093	4,372	4	3	3	75,00%	75,00%
Muestra 18	250	1166	4,664	3	2	2	66,67%	66,67%
Muestra 19	250	1112	4,448	4	3	3	75,00%	75,00%
Muestra 20	250	1148	4,592	4	3	3	75,00%	75,00%
Muestra 21	250	1208	4,832	4	1	2	25,00%	50,00%
Muestra 22	250	1211	4,844	5	2	2	40,00%	40,00%
Muestra 23	250	1214	4,856	4	3	3	75,00%	75,00%
Muestra 24	250	1212	4,848	4	2	3	50,00%	75,00%
Muestra 25	250	1218	4,872	3	2	2	66,67%	66,67%
Muestra 26	250	1117	4,468	4	3	3	75,00%	75,00%
Muestra 27	250	1176	4,704	4	2	2	50,00%	50,00%
Muestra 28	250	1161	4,644	4	3	3	75,00%	75,00%
Muestra 29	250	1134	4,536	5	3	3	60,00%	60,00%
Muestra 30	250	1170	4,680	4	1	1	25,00%	25,00%
	250	**1169,03333**	**4,676**	**3,866666667**	**2,2**	**2,266666667**	**57,22%**	**58,89%**
							DIFF	**1,67%**

Figura D.13: Datos para una red de 250 nodos con una probabilidad de enlace de 0.0375

P = 0.05	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	250	1538	6,15	3	2	2	66,67%	66,67%
Muestra 2	250	1528	6,11	4	2	2	50,00%	50,00%
Muestra 3	250	1537	6,15	3	2	2	66,67%	66,67%
Muestra 4	250	1528	6,11	4	2	3	50,00%	75,00%
Muestra 5	250	1556	6,22	4	2	2	50,00%	50,00%
Muestra 6	250	1514	6,06	4	2	2	50,00%	50,00%
Muestra 7	250	1571	6,28	4	3	3	75,00%	75,00%
Muestra 8	250	1577	6,31	4	2	2	50,00%	50,00%
Muestra 9	250	1583	6,33	4	2	3	50,00%	75,00%
Muestra 10	250	1546	6,18	4	2	2	50,00%	50,00%
Muestra 11	250	1528	6,11	3	1	1	33,33%	33,33%
Muestra 12	250	1524	6,10	4	3	3	75,00%	75,00%
Muestra 13	250	1531	6,12	4	2	3	50,00%	75,00%
Muestra 14	250	1541	6,16	4	2	2	50,00%	50,00%
Muestra 15	250	1551	6,20	4	2	2	50,00%	50,00%
Muestra 16	250	1532	6,13	3	2	2	66,67%	66,67%
Muestra 17	250	1540	6,16	4	2	2	50,00%	50,00%
Muestra 18	250	1577	6,31	3	2	2	66,67%	66,67%
Muestra 19	250	1511	6,04	4	3	3	75,00%	75,00%
Muestra 20	250	1595	6,38	4	3	3	75,00%	75,00%
Muestra 21	250	1557	6,23	4	2	3	50,00%	75,00%
Muestra 22	250	1568	6,27	4	2	3	50,00%	75,00%
Muestra 23	250	1618	6,47	4	3	3	75,00%	75,00%
Muestra 24	250	1555	6,22	3	1	1	33,33%	33,33%
Muestra 25	250	1535	6,14	3	2	2	66,67%	66,67%
Muestra 26	250	1607	6,43	4	2	2	50,00%	50,00%
Muestra 27	250	1553	6,21	4	2	3	50,00%	75,00%
Muestra 28	250	1594	6,38	3	2	2	66,67%	66,67%
Muestra 29	250	1584	6,34	4	2	2	50,00%	50,00%
Muestra 30	250	1580	6,32	4	3	3	75,00%	75,00%
			6,2212	**3,733333333**			**57,22%**	**62,22%**
							DIFF	**5,00%**

Figura D.14: Datos para una red de 250 nodos con una probabilidad de enlace de 0.05

P = 0.0725	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	250	2248	8,992	4	1	1	25,00%	25,00%
Muestra 2	250	2220	8,880	4	2	2	50,00%	50,00%
Muestra 3	250	2188	8,752	4	3	3	75,00%	75,00%
Muestra 4	250	2344	9,376	4	1	2	25,00%	50,00%
Muestra 5	250	2205	8,820	4	2	2	50,00%	50,00%
Muestra 6	250	2308	9,232	3	2	2	66,67%	66,67%
Muestra 7	250	2314	9,256	3	2	2	66,67%	66,67%
Muestra 8	250	2247	8,988	3	2	2	66,67%	66,67%
Muestra 9	250	2238	8,952	3	2	2	66,67%	66,67%
Muestra 10	250	2256	9,024	3	2	2	66,67%	66,67%
Muestra 11	250	2241	8,964	3	2	2	66,67%	66,67%
Muestra 12	250	2302	9,208	3	2	2	66,67%	66,67%
Muestra 13	250	2302	9,208	3	2	2	66,67%	66,67%
Muestra 14	250	2291	9,164	4	2	3	50,00%	75,00%
Muestra 15	250	2273	9,092	3	2	2	66,67%	66,67%
Muestra 16	250	2355	9,420	3	2	2	66,67%	66,67%
Muestra 17	250	2188	8,752	3	2	2	66,67%	66,67%
Muestra 18	250	2268	9,072	3	2	2	66,67%	66,67%
Muestra 19	250	2253	9,012	4	2	2	50,00%	50,00%
Muestra 20	250	2275	9,100	3	2	2	66,67%	66,67%
Muestra 21	250	2331	9,324	3	2	2	66,67%	66,67%
Muestra 22	250	2288	9,152	3	2	2	66,67%	66,67%
Muestra 23	250	2284	9,136	3	2	2	66,67%	66,67%
Muestra 24	250	2295	9,180	3	2	2	66,67%	66,67%
Muestra 25	250	2287	9,148	3	1	1	33,33%	33,33%
Muestra 26	250	2279	9,116	3	2	2	66,67%	66,67%
Muestra 27	250	2304	9,216	4	2	2	50,00%	50,00%
Muestra 28	250	2215	8,860	4	2	2	50,00%	50,00%
Muestra 29	250	2205	8,820	3	2	2	66,67%	66,67%
Muestra 30	250	2242	8,968	3	2	2	66,67%	66,67%
	250	**2268,2**	**9,073**	**3,3**	**1,933333333**	**2**	**59,72%**	**61,39%**
							DIFF	**1,67%**

Figura D.15: Datos para una red de 250 nodos con una probabilidad de enlace de 0.0725

P = 0.1	Nº Nodos	Nº Edges	dice de malla	Path Original	dos dif en B	os dif en Dijs	% BFS	% Dijkstra
Muestra 1	250	3182	12,728	3	2	2	66,67%	66,67%
Muestra 2	250	3193	12,772	3	2	2	66,67%	66,67%
Muestra 3	250	3105	12,420	3	2	2	66,67%	66,67%
Muestra 4	250	3030	12,120	4	2	2	50,00%	50,00%
Muestra 5	250	3085	12,340	3	2	2	66,67%	66,67%
Muestra 6	250	3070	12,280	3	2	2	66,67%	66,67%
Muestra 7	250	3117	12,468	3	2	2	66,67%	66,67%
Muestra 8	250	3120	12,480	3	2	2	66,67%	66,67%
Muestra 9	250	3159	12,636	3	2	2	66,67%	66,67%
Muestra 10	250	3184	12,736	3	1	1	33,33%	33,33%
Muestra 11	250	3135	12,540	3	1	1	33,33%	33,33%
Muestra 12	250	3201	12,804	3	2	2	66,67%	66,67%
Muestra 13	250	3091	12,364	3	1	1	33,33%	33,33%
Muestra 14	250	3141	12,564	3	1	2	33,33%	66,67%
Muestra 15	250	3142	12,568	3	2	2	66,67%	66,67%
Muestra 16	250	3092	12,368	3	2	2	66,67%	66,67%
Muestra 17	250	3130	12,520	3	2	2	66,67%	66,67%
Muestra 18	250	3149	12,596	3	1	1	33,33%	33,33%
Muestra 19	250	3061	12,244	3	1	1	33,33%	33,33%
Muestra 20	250	3086	12,344	3	2	2	66,67%	66,67%
Muestra 21	250	3180	12,720	3	2	2	66,67%	66,67%
Muestra 22	250	3053	12,212	3	2	2	66,67%	66,67%
Muestra 23	250	3045	12,180	3	2	2	66,67%	66,67%
Muestra 24	250	3170	12,680	3	2	2	66,67%	66,67%
Muestra 25	250	3063	12,252	3	2	2	66,67%	66,67%
Muestra 26	250	3029	12,116	3	2	2	66,67%	66,67%
Muestra 27	250	3077	12,308	3	2	2	66,67%	66,67%
Muestra 28	250	3078	12,312	3	2	2	66,67%	66,67%
Muestra 29	250	3087	12,348	3	2	2	66,67%	66,67%
Muestra 30	250	3102	12,408	3	2	2	66,67%	66,67%
			12,448	**3,03333333**			**59,44%**	**60,56%**
							DIFF	**1,11%**

Figura D.16: Datos para una red de 250 nodos con una probabilidad de enlace de 0.1

Para las redes de **300** nodos:

P = 0.025	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	300	1099	3,663	5	3	3	60,00%	60,00%
Muestra 2	300	1132	3,773	4	2	2	50,00%	50,00%
Muestra 3	300	1167	3,890	5	3	4	60,00%	80,00%
Muestra 4	300	1091	3,637	4	1	1	25,00%	25,00%
Muestra 5	300	1186	3,953	4	2	3	50,00%	75,00%
Muestra 6	300	1124	3,747	4	2	2	50,00%	50,00%
Muestra 7	300	1120	3,733	4	2	2	50,00%	50,00%
Muestra 8	300	1082	3,607	5	3	4	60,00%	80,00%
Muestra 9	300	1137	3,790	5	3	3	60,00%	60,00%
Muestra 10	300	1145	3,817	3	2	2	66,67%	66,67%
Muestra 11	300	1126	3,753	4	2	3	50,00%	75,00%
Muestra 12	300	1184	3,947	3	2	2	66,67%	66,67%
Muestra 13	300	1120	3,733	4	2	2	50,00%	50,00%
Muestra 14	300	1127	3,757	3	2	2	66,67%	66,67%
Muestra 15	300	1087	3,623	5	3	3	60,00%	60,00%
Muestra 16	300	1083	3,610	5	3	3	60,00%	60,00%
Muestra 17	300	1147	3,823	5	3	4	60,00%	80,00%
Muestra 18	300	1122	3,740	4	2	2	50,00%	50,00%
Muestra 19	300	1203	4,010	5	2	2	40,00%	40,00%
Muestra 20	300	1120	3,733	4	2	2	50,00%	50,00%
Muestra 21	300	1064	3,547	4	3	3	75,00%	75,00%
Muestra 22	300	1120	3,733	5	2	2	40,00%	40,00%
Muestra 23	300	1113	3,710	4	3	3	75,00%	75,00%
Muestra 24	300	1117	3,723	5	3	3	60,00%	60,00%
Muestra 25	300	1047	3,490	5	2	2	40,00%	40,00%
Muestra 26	300	1165	3,883	4	3	3	75,00%	75,00%
Muestra 27	300	1200	4,000	4	2	2	50,00%	50,00%
Muestra 28	300	1144	3,813	4	3	3	75,00%	75,00%
Muestra 29	300	1092	3,640	4	3	3	75,00%	75,00%
Muestra 30	300	1155	3,850	5	2	2	40,00%	40,00%
	200	**1127,3**	**3,758**	**4,3**	**2,4**	**2,566666667**	**56,33%**	**60,00%**
							DIFF	**3,67%**

Figura D.17: Datos para una red de 300 nodos con una probabilidad de enlace de 0.025

P = 0.0375	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	300	1710	5,700	4	3	3	75,00%	75,00%
Muestra 2	300	1676	5,587	5	2	2	40,00%	40,00%
Muestra 3	300	1575	5,250	4	2	2	50,00%	50,00%
Muestra 4	300	1690	5,633	4	1	1	25,00%	25,00%
Muestra 5	300	1608	5,360	4	3	3	75,00%	75,00%
Muestra 6	300	1669	5,563	4	2	2	50,00%	50,00%
Muestra 7	300	1657	5,523	4	3	3	75,00%	75,00%
Muestra 8	300	1717	5,723	4	2	2	50,00%	50,00%
Muestra 9	300	1611	5,370	4	2	2	50,00%	50,00%
Muestra 10	300	1633	5,443	4	2	2	50,00%	50,00%
Muestra 11	300	1681	5,603	4	3	3	75,00%	75,00%
Muestra 12	300	1769	5,897	4	2	2	50,00%	50,00%
Muestra 13	300	1665	5,550	3	2	2	66,67%	66,67%
Muestra 14	300	1667	5,557	4	3	3	75,00%	75,00%
Muestra 15	300	1677	5,590	4	2	2	50,00%	50,00%
Muestra 16	300	1658	5,527	5	3	3	60,00%	60,00%
Muestra 17	300	1684	5,613	4	2	2	50,00%	50,00%
Muestra 18	300	1704	5,680	4	3	3	75,00%	75,00%
Muestra 19	300	1674	5,580	3	2	2	66,67%	66,67%
Muestra 20	300	1682	5,607	4	3	3	75,00%	75,00%
Muestra 21	300	1729	5,763	3	1	1	33,33%	33,33%
Muestra 22	300	1620	5,400	3	2	2	66,67%	66,67%
Muestra 23	300	1728	5,760	4	2	3	50,00%	75,00%
Muestra 24	300	1712	5,707	4	3	3	75,00%	75,00%
Muestra 25	300	1621	5,403	4	3	3	75,00%	75,00%
Muestra 26	300	1670	5,567	3	1	1	33,33%	33,33%
Muestra 27	300	1739	5,797	3	2	2	66,67%	66,67%
Muestra 28	300	1632	5,440	3	2	2	66,67%	66,67%
Muestra 29	300	1680	5,600	3	2	2	66,67%	66,67%
Muestra 30	300	1668	5,560	4	1	1	25,00%	25,00%
	200	1673,53333	5,578	3,8	2,2	2,233333333	58,06%	58,89%
							DIFF	0,83%

Figura D.18: Datos para una red de 300 nodos con una probabilidad de enlace de 0.0375

P = 0.05	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	300	2157	7,19	3	2	2	66,67%	66,67%
Muestra 2	300	2308	7,69	3	2	2	66,67%	66,67%
Muestra 3	300	2130	7,10	4	3	3	75,00%	75,00%
Muestra 4	300	2218	7,39	4	2	3	50,00%	75,00%
Muestra 5	300	2250	7,50	3	2	2	66,67%	66,67%
Muestra 6	300	2194	7,31	3	2	2	66,67%	66,67%
Muestra 7	300	2291	7,64	4	3	3	75,00%	75,00%
Muestra 8	300	2297	7,66	3	2	2	66,67%	66,67%
Muestra 9	300	2239	7,46	3	2	2	66,67%	66,67%
Muestra 10	300	2241	7,47	3	2	2	66,67%	66,67%
Muestra 11	300	2265	7,55	4	2	2	50,00%	50,00%
Muestra 12	300	2212	7,37	2	1	1	50,00%	50,00%
Muestra 13	300	2198	7,33	4	1	1	25,00%	25,00%
Muestra 14	300	2187	7,29	3	2	2	66,67%	66,67%
Muestra 15	300	2286	7,62	4	2	2	50,00%	50,00%
Muestra 16	300	2264	7,55	4	2	3	50,00%	75,00%
Muestra 17	300	2220	7,40	3	2	2	66,67%	66,67%
Muestra 18	300	2261	7,54	4	2	2	50,00%	50,00%
Muestra 19	300	2301	7,67	3	2	2	66,67%	66,67%
Muestra 20	300	2198	7,33	3	2	2	66,67%	66,67%
Muestra 21	300	2214	7,38	3	2	2	66,67%	66,67%
Muestra 22	300	2239	7,46	3	1	1	33,33%	33,33%
Muestra 23	300	2212	7,37	4	3	3	75,00%	75,00%
Muestra 24	300	2287	7,62	4	2	2	50,00%	50,00%
Muestra 25	300	2247	7,49	4	2	3	50,00%	75,00%
Muestra 26	300	2267	7,56	4	2	2	50,00%	50,00%
Muestra 27	300	2211	7,37	4	1	3	25,00%	75,00%
Muestra 28	300	2242	7,47	4	3	3	75,00%	75,00%
Muestra 29	300	2177	7,26	4	3	3	75,00%	75,00%
Muestra 30	300	2225	7,42	4	1	1	25,00%	25,00%
			7,448666667	3,5			57,78%	61,94%
							DIFF	4,17%

Figura D.19: Datos para una red de 300 nodos con una probabilidad de enlace de 0.05

P = 0.0725	Nº Nodos	Nº Edges	índice de mallado	Path Original	Nodos dif en BFS	Nodos dif en Dijsktra	% BFS	% Dijkstra
Muestra 1	300	3365	11,217	3	2	2	66,67%	66,67%
Muestra 2	300	3252	10,840	3	1	1	33,33%	33,33%
Muestra 3	300	3177	10,590	2	1	1	50,00%	50,00%
Muestra 4	300	3270	10,900	3	2	2	66,67%	66,67%
Muestra 5	300	3263	10,877	3	2	2	66,67%	66,67%
Muestra 6	300	3261	10,870	3	2	2	66,67%	66,67%
Muestra 7	300	3373	11,243	3	2	2	66,67%	66,67%
Muestra 8	300	3249	10,830	3	2	2	66,67%	66,67%
Muestra 9	300	3298	10,993	3	2	2	66,67%	66,67%
Muestra 10	300	3146	10,487	3	1	1	33,33%	33,33%
Muestra 11	300	3238	10,793	3	2	2	66,67%	66,67%
Muestra 12	300	3339	11,130	3	2	2	66,67%	66,67%
Muestra 13	300	3148	10,493	3	1	1	33,33%	33,33%
Muestra 14	300	3298	10,993	3	2	2	66,67%	66,67%
Muestra 15	300	3255	10,850	3	2	2	66,67%	66,67%
Muestra 16	300	3235	10,783	4	2	2	50,00%	50,00%
Muestra 17	300	3286	10,953	3	2	2	66,67%	66,67%
Muestra 18	300	3311	11,037	3	2	2	66,67%	66,67%
Muestra 19	300	3209	10,697	3	2	2	66,67%	66,67%
Muestra 20	300	3278	10,927	4	2	2	50,00%	50,00%
Muestra 21	300	3274	10,913	3	2	2	66,67%	66,67%
Muestra 22	300	3314	11,047	4	2	2	50,00%	50,00%
Muestra 23	300	3301	11,003	3	2	2	66,67%	66,67%
Muestra 24	300	3266	10,887	3	2	2	66,67%	66,67%
Muestra 25	300	3256	10,853	4	2	3	50,00%	75,00%
Muestra 26	300	3261	10,870	4	1	1	25,00%	25,00%
Muestra 27	300	3253	10,843	4	2	2	50,00%	50,00%
Muestra 28	300	3288	10,960	3	2	2	66,67%	66,67%
Muestra 29	300	3172	10,573	3	2	2	66,67%	66,67%
Muestra 30	300	3245	10,817	3	2	2	66,67%	66,67%
	200	3262,7	10,876	3,166666667	1,833333333	1,866666667	58,61%	59,44%
							DIFF	0,83%

Figura D.20: Datos para una red de 300 nodos con una probabilidad de enlace de 0.0725

P = 0.1	Nº Nodos	Nº Edges	dice de malla	Path Originalo	dos dif en B	os dif en Dijs	% BFS	% Dijkstra
Muestra 1	300	4531	15,103	3	2	2	66,67%	66,67%
Muestra 2	300	4444	14,813	3	2	2	66,67%	66,67%
Muestra 3	300	4537	15,123	3	2	2	66,67%	66,67%
Muestra 4	300	4417	14,723	3	1	1	33,33%	33,33%
Muestra 5	300	4444	14,813	3	2	2	66,67%	66,67%
Muestra 6	300	4453	14,843	3	2	2	66,67%	66,67%
Muestra 7	300	4439	14,797	3	1	1	33,33%	33,33%
Muestra 8	300	4474	14,913	4	2	2	50,00%	50,00%
Muestra 9	300	4564	15,213	3	2	2	66,67%	66,67%
Muestra 10	300	4412	14,707	3	2	2	66,67%	66,67%
Muestra 11	300	4355	14,517	3	2	2	66,67%	66,67%
Muestra 12	300	4425	14,750	3	2	2	66,67%	66,67%
Muestra 13	300	4425	14,750	3	1	1	33,33%	33,33%
Muestra 14	300	4513	15,043	3	2	2	66,67%	66,67%
Muestra 15	300	4509	15,030	3	2	2	66,67%	66,67%
Muestra 16	300	4628	15,427	3	2	2	66,67%	66,67%
Muestra 17	300	4467	14,890	3	2	2	66,67%	66,67%
Muestra 18	300	4428	14,760	3	2	2	66,67%	66,67%
Muestra 19	300	4547	15,157	3	2	2	66,67%	66,67%
Muestra 20	300	4592	15,307	3	2	2	66,67%	66,67%
Muestra 21	300	4460	14,867	3	2	2	66,67%	66,67%
Muestra 22	300	4576	15,253	3	2	2	66,67%	66,67%
Muestra 23	300	4377	14,590	3	2	2	66,67%	66,67%
Muestra 24	300	4539	15,130	3	2	2	66,67%	66,67%
Muestra 25	300	4532	15,107	3	2	2	66,67%	66,67%
Muestra 26	300	4571	15,237	4	2	2	50,00%	50,00%
Muestra 27	300	4451	14,837	4	2	2	50,00%	50,00%
Muestra 28	300	4447	14,823	3	2	2	66,67%	66,67%
Muestra 29	300	4401	14,670	3	1	1	33,33%	33,33%
Muestra 30	300	4460	14,867	3	2	2	66,67%	66,67%
			14,935	3,1			60,56%	60,56%
							DIFF	0,00%

Figura D.21: Datos para una red de 300 nodos con una probabilidad de enlace de 0.1